ibsurgeon

- en
- br
- Ru

Search

IB Surgeon Toggle navigation

- Home
- Contacts
- Customers
- Services
  - Firebird/Interbase Recovery
  - Database performance optimization
  - Optimized Firebird configurations
  - Firebird database migration
- Purchase
- Products
  - HQbird: Advanced Firebird SQL
  - IBSurgeon FirstAID 4.0
  - IBUndelete 2.6
  - Upgrades
  - Previous versions
    - IBSurgeon ISV Subscription
    - IBSurgeon Recovery Pack
    - FBScanner
    - IBAnalyst
    - FBDataGuard
    - FBMonLogger
    - IBTM Transaction Monitor
    - IBBackupSurgeon
    - IBSurgeon Free Tools
- Articles
- Tests
  - Firebird 2.5 Performance Logs
  - Firebird 3.0 Performance Logs
  - Firebird 4.0 Performance Logs
- Documentation
- Firebird Training

Articles

1. Home
2. Articles
3. 45 Ways To Speed Up Firebird Database

45 Ways To Speed Up Firebird Database

Here you can find the list of performance tips for Firebird database in different areas - from hardware/OS and Firebird configuration tuning to SQL optimization recommendations. This list is not the complete reference how to optimize Firebird, and it assumes that you understand basics of Firebird functioning, such as execution plans, transactions management, and queries performance statistics.
Please apply these tips with caution and verify their effect before putting to the production.
Our company (IBSurgeon) offers the comprehensive database performance optimization service and Firebird training through Skype.

1. Put database to SSD

Put your database on SSD. SSD drive provides much better random IO than traditional drives. Random IO is critical for reading and writing data distributed through big database file - the majority of database operations require intensive parallel random IO.

2. Use RAID 10

If you use RAID1 or RAID5, consider RAID10 – it is 15-25% faster.

3. Check BBU

If you are using RAID controller, check that it has Backup Battery Unit (BBU) installed and operational – some vendors do not provide BBU by default. Without BBU, the controller disables the cache, and RAID works very slow, even slower than usual SATA drives. Usually, you can check BBU status in the RAID configuration tool.

4. Set write cache to write-back

If you are using RAID controller with installed BBU (and server with UPS), check that its cache is set to write-back (not write-through). «Write-back» enables write cache of the controller.

5. Enable read cache

If you use RAID controller, check that it has enabled read cache.

6. Check disk subsystem

Check your drives for bad blocks and other hardware problems (including overheating). Hardware problems can significantly decrease IO performance and lead to database corruptions.

7. Use SuperClassic or Classic in Firebird 2.5

If you use Firebird 2.5 SuperServer with many connections, try to use SuperClassic or Classic, they can scale better by using all cores of CPU.

8. Use SuperServer 3.0 in Firebird 3.

If you use Classic or SuperClassic in 2.5, consider migration to Firebird 3.0 SuperServer, now it can use multiple cores and combine it with the advantages of the shared cache.

9. Increase page buffers cache

Increase the size of page buffers cache (parameter DefaultDBCachePages) from the default values. For 2.5 SuperServer we recommend 10000 pages, for 3.0 SuperServer – 50000 pages, for Classic and SuperClassic – from 256 to 2048 pages. However, don't set page buffers cache value too high – cache synchronization has its cost, and the idea to put all database into RAM by tuning this value will not work. Use pre-optimized Firebird configuration files here: http://ib-aid.com/en/optimized-firebird-configuration/

10. Increase memory size for sort operations

Increase the value of TempCacheLimit parameter in firebird.conf – it specifies the size of the cache of the temporary space for sorting. Default values are too low (8Mb for Classic and 64Mb for SuperServer), use at least 64Mb for Classic and 1Gb for SuperServer and SuperClassic. Again, use optimized configuration files from #9.

11. Set Forced Writes Off (with caution!)

If you have intensive insert or update activity (you can check it with HQbird MonLogger, for details see page 60 of HQbird User Guide), and if you have UPS and replication installed to protect from hardware failures, consider to set Forced Writes settings to OFF, it can increase speed of write operations up to 3 times.

12. Increase number of hash slots for Classic/SuperClassic

Increase the value of LockHashSlots parameter for Classic and SuperClassic from the default 1009 to some big prime number (30011, for example), it will decrease queues in the internal locking mechanism.

13. Use CPU Affinity for Super Server 2.5

If you use SuperServer 2.5, set CPUAffinity parameter to the value equal to the number of databases in use: SuperServer in 2.5 can use different CPU cores to process requests for the certain databases.

14. Use fast drive for temp space

Set the first part of TempDirectory parameter in firebird.conf to the fast disk – SSD or RAM drive. It will decrease the time of big sortings – for example when the database is being restored.

15. Store database backups on another drive

Store database backups on the dedicated physical drive (RAID). It will separate read and write IO during backup, and increase backup speed and decrease load for the main drive. It is especially important when backups are taken while users are working with the database. More details about hardware configuration for Firebird can be found in "Firebird Hardware Guide".

16. Deactivate indices for bulk inserts

If you insert or update many records (more than 25% of the table), deactivate indices for the table where records are inserted and reactivate them after insert or update. The index rebuild operation can be faster

than many updates of the index.

17. Use Global Temporary Tables for fast inserts

To speed up inserts and updates, use Global Temporary Tables for bulk inserts of the large recordsets, and then transfer records into the permanent table. It can be very effective to insert records to GTT, pre-process them and then move to the persistent table.

18. Avoid unnecessary indices

Use fewer indices for tables with intensive inserts and updates. Each index adds significant overhead for insert, update, delete, and garbage collection operations – there could be 3-4 additional page reads and writes when the single record is being inserted/updated/deleted/cleaned for each index.

19. Replace UDFs with embedded functions calls

Replace UDF calls with embedded functions calls. Many embedded functions were added in the recent versions of Firebird, which offer functionality previously available only in UDF libraries. Replace such functions where possible, since embedded functions work up to 3 times faster than UDFs.

20. Use read-only transactions for read operations

Use read-only transactions for operations which do not change record (i.e., SELECTs) with isolation mode = read committed. Such transactions do not retain record versions from the garbage collection, and can run indefinitely: they do not affect database performance.

21. Use short write transactions and get rid of ALL long-running

Use short writeable transactions (for operations INSERT/UPDATE/DELETE).
The shorter writeable transaction is, the better. The short transactions retain proportionally less number of record versions from garbage collection than long-running. Unfortunately, even the single long-running transaction (from the development tool left open, for example) can screw the good effect of all other short writeable transaction. That's why you need to monitor long-running transaction and fix the appropriate places in the source code. Use HQbird DataGuard tool to receive alerts about the oldest active transaction in Firebird database (what applications started it, what IP address, the timestamp of its start), and HQbird MonLogger tool to see the complete list of the long-running active transactions and their IO statistics. Also, if you are using database access components/libraries which can cache record sets, use cached updates.

22. Avoid long record chains

Avoid situations when one record has many record versions – Firebird works much slower with long record chains. (to see how many record versions some tables has, and what is the longest record chain you can use HQbird IBAnalyst tool, tab Tables, sort on "Max Version"). Use the combination of inserts and scheduled delete of old records instead of multiple updates of the same record.

23. Use PREPARE correctly

Use prepared statements to run SQL queries where only parameters are changed – for example, make prepare before the loop of such queries. Prepare can take significant time (especially for big tables), and preparing the query only once will greatly increase the overall performance.

24. Don't COMMIT too often during bulk insert/update operation

In the case of bulk INSERT/UPDATE/DELETE operation, don't commit the transaction after each change (it can happen if you are using auto commit option in your database driver) - commit transactions at least after 1000 operations or more. Each transaction commit runs several read/write IO operations against the database, that's why often commits decrease database performance.

25. "Turn off" indices if you are using IN with many constants

If you are using construction WHERE fieldX IN (Constant1, Constant2,… ConstantN), and there is an index on fieldX, Firebird will use an index as many times as many constants are in the IN list. Disable index search by turning fieldX into expression +0: WHERE fieldX+0 IN (Constant1, Constant2,… ConstantN), or, for strings, use fieldX||''

26. Replace IN with JOIN

Avoid using queries with nested WHERE IN(SELECT... WHERE IN (SELECT.. WHERE IN())), it can confuse Firebird optimizer. Transform nested INs into joins.

27. Use LEFT JOIN in the correct way

If you are using LEFT OUTER joins, explicitly put tables in the join from the smallest one to the largest one.

28. Limit fetch of SELECT queries

Always try to limit the large output for SELECT queries with FIRST… SKIP or ROWS clauses. If the query is not designed specifically as a report (which requires all records to be printed/exported), usually it is enough to show top 10-100 records. Fetch only necessary records.

29. Specify less number of columns in SELECT with ORDER BY/GROUP BY

Reduce the number of columns and their summary width in queries with ORDER BY/GROUP BY both in SELECT part (i.e., fields to be shown) and in the ORDER BY clause. Firebird merges columns from SELECT and ORDER BY/GROUP BY clauses and sorts them in memory (or, if memory is not enough, on the disk). So, if there is a long VARCHAR in SELECT, the size of the sort files can be really large (many gigabytes). Reducing the number of fields only for those which must be sorted and late join with big fields to be shown can greatly (x3-x10) increase speed of a query with ORDER BY/GROUP BY.

30. Use derived tables to optimize SELECT with ORDER BY/GROUP BY

Another way to optimize SQL query with sorting is to use derived tables to avoid unnecessary sort operations. Instead of

```
SELECT FIELD_KEY, FIELD1, FIELD2, ... FIELD_N
FROM T
ORDER BY FIELD2
```

use the following modification:

```
SELECT T.FIELD_KEY, T.FIELD1, T.FIELD2, ... T.FIELD_N
FROM (SELECT FIELD_KEY FROM T ORDER BY FIELD2) T2
JOIN T ON T.FIELD_KEY = T2.FIELD_KEY
```

31. Store short strings in VARCHAR, large in BLOBs

To store short character data, use VARCHARs, to store long texts, use BLOBs. Varchars are faster for the small pieces of data because they are stored in the record, and the whole record is read during the same IO cycle, and if record size is less than 2/3 of the database page size, the whole record is stored on the same database page. BLOBs are stored outside of the record, and require the additional round of IO to read it, and they show the advantage with reading and writing long strings.

32. Exclude BLOB columns from the large SELECTs

Exclude BLOB columns from the large SELECTs. Use a kind of late binding with sub-selects to selectively show information from BLOBs (for example, show the content of the document).

33. Use BIGINT for primary and unique keys

Use BIGINT type for auto-incremented primary and unique keys and for identifiers of all types. Operations with BIGINT are the fastest, and BIGINT has enough capacity to store almost all data ranges.

34. Don't use VARCHARs for keys

Don't use VARCHAR for identifiers unless it is really necessary – operations with them are far less effective than with integer columns. Especially avoid GUIDs, as identifies – due to the random distribution of GUID values INSERT/UPDATE operations with Primary/Unique Keys GUIDs can be 20 times slower than with integers.

35. Recalculate indices statistics

Recalculate indices statistics regularly. Update indices statistics for the tables with frequent or massive changes with command SET STATISTICS, it allows Firebird optimizer to choose better SQL plans. HQbird Firebird DataGuard can perform such recalculation of indices statistics automatically according to the desired schedule (usually once a week).

36. Use connection pool

If database connections to Firebird database are short (it's typical for websites), use connection pool – for example, in PHP use function ibase_pconnect instead of ibase_connect

37. Use LINGER option in Firebird 3.0

If database connections are short and you are using Firebird 3+, use LINGER option to keep cache active during the specified amount of time, it will keep frequently used pages in the cache even if there will be no other connections. For example, ALTER DATABASE SET LINGER TO 60 will keep the cache for 60 seconds after the end of the last connection.

38. Use HASH JOINs

In Firebird 3.0, in case the of joining big and small tables, HASH JOIN could be much faster than normal join which uses «nested loop» with index. To make Firebird optimizer to use HASH join, use +0 in the join condition: T1 JOIN T2 ON T1.FIELD1+0 = T2.FIELD2+0. Check the result of the optimization before putting it to the production!

39. Mark appropriate PSQL functions as DETERMINISTIC

Mark your PSQL functions (in Firebird 3+) which do not have parameters and return constant values with keyword DETERMINISTIC. The deterministic functions are calculated and cached in the scope of the current query.

40. Use analytical (window) functions in Firebird 3.0

If you are running SELECT with simultaneous output of some column and aggregated function for it, use window (analytical) functions – it is faster than the subquery or 2 queries. For example:

```
Select id, department, salary, salary / (select sum(salary) from employee) percentage
from employee
```

replace with

```
Select id, department, salary, salary / sum(salary) OVER () percentage from employee
```

41. Use switch -se for gbak

Use switch –se to increase gbak backup and/or restore speed up to 20%, for example

```
gbak -b -g -se service_mgr c:\db\data.fdb e:\backup\data.fbk
```

42. WHERE CURRENT OF

The fastest way to process records fetched by the cursor in PSQL is the clause ‘where current of <>’. It is faster than ‘where rb$db_key = :v_db_key’ and much faster than search with a primary or unique key.

43. Avoid often queries to monitoring tables

Don't run queries to Firebird monitoring tables (MON$) too often – such queries consume significant resources and can greatly decrease the performance of the main business logic. We recommend running MON$ queries not often than once per minute. For continuous monitoring of Firebird queries/transactions /attachments, use HQbird PerfMon tool which supports Trace API (see page 66 of HQbird User Guide for details) .

44. Use NO_AUTO_UNDO option for bulk inserts/updates

If you are running many DML (Update/Insert/Delete) commands in the frames of the same transaction, Firebird merges undo-log of each command with undo-log of the transaction. To speed up bulk DML operations start the transaction with «NO AUTO UNDO» option, in order to do not merge undo-logs of each command with the transaction's undo-log.

45. Do not use SRP authentication in Firebird 3 if you don't need it

Does not use SRP users authentication (Firebird 3.0+) if you don't really need it – connect with SRP authentication is established slower than the regular connection.

Instead of summary

The biggest impact to the Firebird performance is caused by your own SQL queries. Learn how to read

and analyze query plans, this is a key for the queries performance. You can get an online course about queries optimization in IBSurgeon http://ib-aid.com/firebird-training

Contact us

Do you have any questions? Don't hesitate to contact us by email!

In order to receive notification about next 55 ways to improve Firebird performance, subscribe to IBSurgeon news:

email address

Subscribe