

# gstat examples and interpretation

This section contains frequently executed statistics gatherings and explains the output.

## Database header

This option produces the least amount of output - unless you specify a single nonexistent table name with the `-t[table]` switch - and is included with all other switches, so it is discussed first.

```
tux> gstat employee -header
```

```
Database "/opt/firebird/examples/empbuild/employee.fdb"
```

```
Database header page information:
```

Flags	0
Checksum	12345
Generation	184
Page size	4096
ODS version	11.1
Oldest transaction	166
Oldest active	167
Oldest snapshot	167
Next transaction	170
Bumped transaction	1
Sequence number	0
Next attachment ID	68
Implementation ID	19
Shadow count	0
Page buffers	0
Next header page	0
Database dialect	3
Creation date	Sep 25, 2009 12:50:24
Attributes	multi-user maintenance

```
Variable *header data:
```

```
Sweep interval: 20000
```

```
*END*
```

The first line of output displays the database filename(s) and path. This can be useful to resolve a database [alias](#) to find out exactly where the database is located. As the employee database is a single-file database, only one file is displayed. Had this been a multiple-file database, the end of the listing above would look like the following:

```
...
```

```
Variable header data:
```

```
Continuation file: /u00/firebird/databases/multi_employee.fdb1
```

```
Last logical page: 162
```

The details of the various header fields are described below:

*Flags:* Flags are not used on a [database header page](#).

*Checksum:* All checksums are 12345. Checksums on the various database pages are no longer used.

*Generation:* The generation number is incremented each and every time this page is rewritten in the database.

*Page size:* The [page size](#) of the entire database. As the database file has to be split into various pages, the SYSDBA can, at creation time, specify how big a page size he or she desires. Every page in the database will be the same size.

*ODS version:* The [On-Disk Structure](#) of a database defines, possibly along with the [SQL dialect](#), which features of the Firebird database system are available to users of that database. These features may be present in the version of Firebird that you are running, but if the database ODS is older, some of the new features will not be available.

Values you may currently see here are:

- 5.0 for InterBase® 3.3
- 8.0 for InterBase® 4.0
- 9.0 for InterBase® 4.5
- 9.1 for InterBase® 5.0
- 10.0 for Firebird 1.0 and InterBase 6.0
- 10.1 for Firebird 1.5
- 11.0 for Firebird 2.0
- 11.1 for Firebird 2.1
- 11.2 for Firebird 2.5

*Transaction details:* There are a number of different [transaction](#) details in the report; these are:

- *Oldest transaction* - the transaction ID of what is known as [Oldest Interesting Transaction](#) or OIT. This is simply the ID of the longest running transaction that has so far not been completed by way of a *hard commit*. It may have been rolled back, or be [in limbo](#), but if it has been committed, it is no longer interesting. This value, along with the Oldest Snapshot Transaction, is used when determining if an automatic sweep of the database is required.

*Note:* There are two commits - commit and commit retaining. Only the first of these is a hard commit, which, when executed renders the transaction as no longer interesting. Commit retaining leaves the transaction as still interesting. Some database utilities and/or tools that commit actually perform a commit retaining which can leave your database with a lot of still interesting transactions.

- *Oldest active* - the ID of the oldest active transaction, or OAT. This value shows the [transaction ID \(TID\)](#) of the oldest transaction that is still running. A transaction is considered active if it has not been hard committed, is not in a state of limbo and has not been rolled back.
- *Oldest Snapshot* - the ID of the oldest transaction which is currently not eligible to be [garbage-collected](#). Any transaction with this or a higher ID cannot, yet, have old record versions removed by a sweep, for example. Normally, this is the same as the OAT above. The difference between *this* value and the [OIT](#), if greater than the database sweep interval - assuming that automatic sweeping is not disabled - determines if an automatic sweep takes place.

*Note:* Many websites, books, manuals (previously including this one) explain that the automatic sweep is activated when OAT - OIT is greater than the sweep interval. This is not the case as explained by Vlad Khorsun, one of the Firebird developers, who explained that it is when OST - OIT is greater than the threshold that the sweep is activated.

- *Next transaction* - The next transaction started on the database will have this ID number.
- *Bumped transaction* - always 1, no longer used.

If you discover that the difference between the *OAT* and the *Next Transaction ID* seems to be growing larger and larger, something in your database is not committing properly and as such, an increasing number of garbage records may be building up. Eventually, you will see that the database startup times take longer and longer and the performance becomes slower and slower. Check the figures and if a problem is detected, you may be wise to run *gfix* to manually run a database sweep to clear out the [garbage](#) and restore normal working to the database.

You may wish to consult with the section entitled [Limbo Transaction Management](#) in the *gfix* manual for details on how to detect and treat transactions in limbo. These may well be affecting the ability of the database sweep process in clearing out old redundant data from older uninteresting transactions. [Limbo transactions](#) are caused when a [two-phase commit](#) across multiple databases, fails for some reason. Limbo transactions are still interesting to the database and need to be committed or rolled back using *gfix* as the sweep processing cannot tell whether or not it is safe to do so without human intervention.

*Sequence number:* Always zero. This was the sequence number of the [database header page](#), but is no longer used.

*Next attachment ID:* The ID number of the next attachment to this database. Every time an application connects to the database, this number goes up by one. Starting up and shutting down the database increases this number too. *gstat* connections do not alter the ID as they do not connect in a normal manner.

*Implementation ID:* When the database was created, it may have been created on a different system - hardware, operating system etc. - to the one on which it is now running. The implementation ID shows you which hardware architecture the database was *originally* created on.

The implementation ID is used to determine if the database can actually be used on the hardware it is currently running on, or if there is some feature of the original hardware, where the database was created, that makes it incompatible with the current host system.

*Shadow count:* Displays the number of [shadow files](#) attached to this database, or available for use by this database. Sometimes this value is incorrect even when shadow files have been created and/or deleted recently.

*Warning:* Because of the inconsistency between what *gstat* reports and reality, it is best to use *isql* and the [SHOW DATABASE](#) command to view correct details of the shadow files.

*Page buffers:* If this value shows as zero, the database is using the server's default value for the number of pages that can be cached in memory when the database is operating. The setting may be defined in the [firebird.conf](#) file. On Firebird Superserver 2.1, this setting is the [DefaultDbCachePages](#) in the configuration file and is set to 2048 pages. You may use *gfix* to change this without editing the configuration file.

*Database dialect:* The database's [SQL dialect](#) number. Normally 1 or 3. This setting can be changed

using [gfix](#) and, alongside the [ODS](#) value, helps determine what features of Firebird are available for use when applications use the database.

*Creation date:* The date that this database was created originally. It may show the date that the database was last restored by [gbak](#).

*Attributes:* This part of the report displays information about various attributes of the database. Examples of what you may see are:

- *no reserve* - All pages will be filled to 100% and will be most useful on read-only databases. No space is reserved in each page for updates and/or deletions.
- *force write* - Disc writes are not cached. They are written out to the hardware at the time of the write request. This is used mainly on Windows databases where the cache management system can lead to lost writes and database corruption.
- *shutdown* - The database has been closed and cannot be used.
- *read only* - The database is running in read-only mode.
- *multi-user maintenance* - The database is closed for maintenance. Multiple connections are allowed by SYSDBA or the database owner only.
- *single-user maintenance* - The database is closed for maintenance. Only one SYSDBA or database owner connection is allowed.

Other values may appear here, depending on the version of Firebird in use and, of course, future releases.

*Variable header data:* This part of the report covers information that is not in the fixed part of the database header. For example, the [sweep interval](#) is displayed here and information applicable to [secondary files](#), if any, that are attached. If you have backed up the database using the [nbackup](#) tool, for example, details of the backup [GUID](#) will be displayed here - but only for the most recent backup.

[back to top of page](#)

## Analyse entire database

The analysis of the entire database is the default for [gstat](#). When used, all user [tables](#) and [indices](#) will be analysed and the gathered statistics reported. As the output will most likely be very large, it is advisable to pipe the output to a file:

```
gstat employee >employee.gst
```

The output will consist of an analysis of each and every user table and all associated user indices. Interpretation of these results is covered below in the sections on analysis of data and index pages.

[back to top of page](#)

# Analyse data pages only

The command to analyse only user [tables](#) in the database is:

```
gstat employee -data >employee.gst
```

And the results output from this command will list the user tables in alphabetical order. No [indices](#) will be analysed or listed regardless of how many may exist within the database.

Once the report has been completed, the results can be analysed as follows, looking at one table in particular.

```
CONFIGREVISIONSTORE (213)
  Primary pointer page: 572, Index root page: 573
  Data pages: 2122, data page slots: 2122, average fill: 82%
  Fill distribution:
    0 - 19% = 1
    20 - 39% = 0
    40 - 59% = 0
    60 - 79% = 79
    80 - 99% = 2042
```

The extract, above, from the report begins by displaying the table name - `CONFIGREVISIONSTORE` - and the table ID - `213`. The table's ID is actually the column `RDB$RELATION_ID` in the system table `RDB$RELATIONS`, as the following [isql](#) session shows:

```
SQL> select rdb$relation_name
CON> from rdb$relations
CON> where rdb$relation_id = 213;
```

```
RDB$RELATION_NAME
=====
CONFIGREVISIONSTORE
```

*Primary pointer page:* This is the page number, within the database, of the first page with pointers to the data pages of this table. The structure of the database is such that each table has exclusive data pages and a list of those pages is required to be kept somewhere. This statistic gives you the page number for that location.

*Index root page:* This is the page number where the first page of pointers to the table's indices can be found within the database. Every table in the database has one page, the index root page, that holds pointers to the apex pages for each individual index.

*Data pages:* The total number of pages allocated to this table. Because `gstat` doesn't connect to the database in a transaction-aware manner, it cannot determine whether any of these pages are old record versions ([garbage](#)) or deleted records in currently uncommitted transactions, so the number may be higher than it needs to be as these additional pages are included in the total.

*Data page slots:* This value should be the same as the number of data pages. It reports on the

number of pointers to pages in this table, that are stored in various pointer pages internal to the database. If the numbers differ, it may be down to the garbage that remains uncollected.

*Average fill:* The calculated space used in each page of the table, on average. The figure includes space utilised by back versions of records in the table. The fill distribution (below) gives more details.

*Fill distribution:* This section of the report displays a 5-band histogram where each band represents 20% of the space filled in each page. In the example above, we see that this table has a single page that is filled less than 20%, 79 pages are filled to between 60% and 79% while the vast majority, 2042, are filled to between 80% and 99%.

[back to top of page](#)

## Analyse index pages only

The command to analyse only user [indices](#) in the database is:

```
gstat employee -index >employee.gst
```

And the results output from this command will list the user [tables](#) in alphabetical order. No tables will be analysed; however, the report will list the table names in alphabetical order and will list all applicable indices beneath the appropriate table name.

Once the analysis has been completed, the results can be interpreted as follows. The following example shows the output from a single index in a database.

```
CONFIGREVISIONSTORE (213)
  Index PK_CONFIGREVISIONSTORE (0)
    Depth: 3, leaf buckets: 174, nodes: 62372
    Average data length: 2.58, total dup: 0, max dup: 0
    Fill distribution:
      0 - 19% = 15
      20 - 39% = 0
      40 - 59% = 55
      60 - 79% = 68
      80 - 99% = 36
```

The above extract from the report begins by displaying the table name - `CONFIGREVISIONSTORE` - and the table ID - `213` as described above.

Following the table's details - and only the name and ID are displayed - the index details are shown. As above, the index name and its ID are displayed. This time, the ID refers to the index's position in the list of all indices created on the table. ID zero is the first index created, ID 1 is the next and so on. The output from [gstat](#) may not list the indices in ID order and if any indices were created but subsequently dropped, there may be gaps in the ID sequence.

The next two lines, after the index name and ID, show the overall statistics for this index.

*Depth:* This statistic displays the number of pages that have to be accessed in order to get at an

index entry. In this example we have to read three separate pages into the buffer cache before we can use the index details to access the row we want in the table. This is often referred to as index indirection.

### Depth: 3

On disc, there is a top level [Index Root Page](#) which is created at the same time as the database. This page holds a list of pointers to the top (apex) page for each index - one page per index. For any given index, this page holds a list of pointers to either:

- another level's apex pages if depth is greater than 1, or,
- to the leaf pages for the actual index data if depth = 1.

The leaf pages store the location of the data that have been indexed. The index depth is the number of levels you have to step down from the index's apex page, to get to the leaf pages. Neither the Index Root Page nor the index's apex page are counted in the depth.

On average, a depth of 2 or less indicates an index that is efficient. If the depth is 3 or more, the index will most likely not be operating at its best. The solution in this situation is to use `gbak` to increase the database [page size](#) by taking a backup and restoring it, as follows:

```
tux> # Shutdown the database
tux> gfix -shut -tran 60 employee

tux> # Backup the database
tux> gbak -backup employee /backups/employee.fbk

tux> # Find current page size
tux> gstat employee -header | grep -i "page size"
      page size          4096

tux> # Restore database with a bigger page size
tux> gbak -replace overwrite -page 8192 /backups/employee.fbk employee

tux> # Check new page size
tux> gstat employee -header | grep -i "page size"
      page size          8192

tux> #Open the database
tux> gfix -online normal employee
```

Once the above has been carried out, you should find that the depth of the index is 2 or less. If this is not the case, simply repeat the process above using an even bigger page size.

**Warning:** The above command to restore the backup *overwrites* the original database file. This works by deleting the original file and recreating it, so you really need to be sure that your database backup actually works and that the *backup* file produced is usable before attempting to overwrite a database. See the [gbak manual](#) for more details.

**Leaf buckets:** This statistic informs us of the number of leaf pages that this particular index uses. A page and a bucket are synonymous but page tends to be the more modern term in wide use.

leaf buckets: 174

In our example index, we see that there are 174 pages in the database holding the details of the indexed values for this table - all of these pages contain pointers to the data.

The number of leaf pages should match up to the sum of the total number of pages in each histogram bar in the fill distribution, shown below.

**Nodes:** This is the total number of records in the table that have been indexed. However, it is possible - because gstat doesn't work in a transaction-aware manner - that this figure will possibly include rows that have been deleted (and not [garbage-collected](#)) and/or it may count records more than once if they have been modified in such a way that the indexed column(s) have been changed.

nodes: 62372

Because of the above, it is advisable to carry out a [sweep](#), or a database [backup](#) & [restore](#), prior to running [gstat](#) to ensure that the statistics gathered are accurate and reflect the true position of the database.

**Average data length:** This statistic indicates the average length of the key column(s) in bytes.

Average data length: 2.58

This is most likely less than the actual sum of the column sizes as Firebird uses index compression to reduce the amount of data held in an index leaf page.

**Duplicates:** Duplicates are not permitted in a [primary key](#) or unique index. Other indexes do permit duplicates and these statistics report on the number of duplicates the index holds. The following isql query shows the details of duplicates for an indexed column in a different table to the one being used so far - which has no duplicates.

```
SQL> SELECT IDX, COUNT(*)
CON> FROM NORMAN_TEST
CON> GROUP BY IDX;
```

IDX	COUNT
1	10
2	4
3	1

From the above we see a total of 15 rows, of which there are 14 duplicated values (all those with a 1 or 2 in the [IDX](#) column). The following is the extract for the duplicates for this table:

Index NORMANX (0)

Depth: 1, leaf buckets: 1, nodes: 15

Average data length: 0.27, total dup: 12, max dup: 9

**Total dup** is the total number of duplicates in the index. Note from the above that only 12 duplicates are listed but we already know that there are 14 duplicate rows in the index. How is this possible?



The first occurrence of a 1 and the first occurrence of a 2 are not counted by `gstat` as duplicates. Only the second and subsequent copies are considered duplicates.

*Note:* In *my opinion* this is not quite correct behaviour. In the table above there are 15 rows and only three unique values in the `IDX` column, which is indexed. My index therefore holds 14 duplicate values rather than just 12.

You can, however, use the `total dup value` to extract the number of unique values in the index by subtracting it from the nodes value.

`Max dup` reports on the number of index entries which share the longest chain of duplicates. In other words - for the above index - there are 9 index entries that share the *same* value in the indexed column. We can see this to be true as the rows where `IDX` is 1 has 9 duplicate entries.

If `max dup` is getting close to `total dup`, then it is a reasonable assumption to conclude that it may be that the index is so poor in selectivity that it may never be used in queries.

*Fill distribution:* The remainder of the report for our original example index shows how the pages are used within the index.

#### Fill distribution:

```
0 - 19% = 15
20 - 39% = 0
40 - 59% = 55
60 - 79% = 68
80 - 99% = 36
```

The figures represent a graph (or histogram) of how the space in the index's pages are being utilised. Each value of the histogram represents the number of pages in the whole index, which have been filled to a certain percentage. Each bar of the histogram represents the percentage filled for the page.

The example index's fill distribution is shown above and from these figures we see that the vast majority of the pages are filled to between 40 and 99%. The individual numbers at the end of each line above show the number of pages in this band. The example shows that:

- 15 pages have been filled to less than 20%; and
- 0 pages have been filled to between 20% and 39%; and
- 55 pages have been filled to between 40% and 59%; and
- 68 pages have been filled to between 60% and 79%; and
- 36 pages are filled to between 80% and 99%.

The sum of all these pages should add up to the same figure shown above for leaf nodes.

This index shows reasonably good space usage as the majority of pages are well filled. Ideally, you would like to see all the pages being filled to between 80 and 99%. If, on the other hand, the report showed that the pages were all lightly filled - say less than 60% - the index would be a good candidate for a rebuild exercise.

Be sure to consider the total number of nodes before starting a rebuild - if there are only a small number of nodes in the index, then rebuilding will not help the space usage as there may not be enough records to actually fill the index pages.

[back to top of page](#)

# Selecting tables to analyse

If you wish to include a specific list of tables in the analysis, rather than all user tables, then you can use the `-table` switch to specify the ones you wish to include. Note that specifying table names in this manner also analyses all indices associated with those tables.

```
gstat employee -t EMPLOYEE JOB COUNTRY >employee.gst
```

The resulting output is interpreted as described above.

If you have a table name that has been created by a user wishing to preserve the letter case of the table name, rather than having it converted to uppercase, for example:

```
tux> isql myMusic
Database:  mymusic

SQL> CREATE TABLE "MyMusic_Artists" (
CON> art_id integer,
CON> art_name ....);

SQL> COMMIT;
```

... then you must supply the table names in double quotes and in exactly the same letter case as the name of the table within the database:

```
gstat mymusic -t "MyMusic_Titles" "MyMusic_Artists" > MyMusic.gst
```

If you supply a non-existing table name, or get the name in the wrong case etc., `gstat` simply ignores it.

[back to top of page](#)

## Including the system tables & indices

Normal use of `gstat` doesn't include the [system tables](#) and [indices](#) in the output. Calling `gstat` with the `-system` switch causes these tables to be included in the analysis.

```
gstat employee -system >employee.gst
```

The interpretation of the results for the various system tables and indices is exactly as described above for user tables and indices.

# Record & version details

When you run `gstat` with either the default switches, or `-d[ata]` or `-t[able]` and add the `-r[ecord]` switch, you get additional information in the report that shows the average record length and average version details for the table(s) in question:

```
Average record length: 96.55, total records: 62372
Average version length: 0.00, total versions: 0, max versions: 0
```

*Average record length:* Simply the average record length, in bytes, of all the records in the table. If this figure is 0.00 then you can be reasonably sure that all your records have been deleted, or that you have no records in the table.

*Total records:* The total number of records in the table. The value may include records in currently active transactions and may include records which have been deleted.

```
tux> # In session 1.
tux> gstat test -r -t NORMAN

...
Analyzing database pages ...
NORMAN (142)
  Primary pointer page: 268, Index root page: 269
  Average record length: 9.00, total records: 15
  Average version length: 0.00, total versions: 0, max versions: 0
  Data pages: 1, data page slots: 1, average fill: 10%
```

```
tux> isql tset -user norman -password secret
Database: employee

SQL> SELECT COUNT(*) FROM NORMAN;

      COUNT
=====
          15
```

At this point, we can see that there are 15 records in the NORMAN table and that the average length of these 15 records is 9.00 bytes. Next, we start another isql session and delete all the records from the NORMAN table.

```
tux> # In session 2.
tux> isql test -user norman -password secret
Database: employee

SQL> DELETE FROM NORMAN;
SQL> COMMIT;
SQL> shell;
```

Still in the second session, we execute `gstat` to fetch statistics for the NORMAN table, the results are

shown below.

```
tux> gstat test -r -t NORMAN

...
Analyzing database pages ...
NORMAN (142)
  Primary pointer page: 268, Index root page: 269
  Average record length: 0.00, total records: 15
  Average version length: 9.00, total versions: 15, max versions: 1
  Data pages: 1, data page slots: 1, average fill: 16%
...

tux> # Return to isql.
tux> exit
```

Comparing the report above with the one taken before we deleted the records, we can see straight away that:

- The average record length indicates that there are no records in the table, but the total record count shows that there are (still) 15. This is a good indicator that a session has deleted all the records but [garbage collection](#) has yet to run.
- The versioning details have all changed, there are now statistics for [average version length](#), [total versions](#) and [max versions](#).
- The average fill for the page(s) in this table has risen from 10% to 16% even though everything has been deleted. The extra space is being used by the back versions of the deleted records.

Continuing in the second session, if we execute a full table scan of the [NORMAN](#) table we will not see any results, but we will garbage collect the back versions.

```
SQL> SELECT * FROM NORMAN;

SQL> shell;

tux> gstat test -r -t NORMAN

...
Analyzing database pages ...
NORMAN (142)
  Primary pointer page: 268, Index root page: 269
  Average record length: 0.00, total records: 0
  Average version length: 0.00, total versions: 0, max versions: 0
  Data pages: 0, data page slots: 0, average fill: 0%
```

Everything has now returned to zero. There are no back versions, no current versions and the page is no longer filled.

*Average version length:* This is similar to the average record length, but for the back versions of the record. For example, if you have deleted a number of records and updated others, the old - back - versions of these records will be reported here. If the figure is 0.00 then garbage collection has taken place and removed the back versions - see above for an example.

*Total versions:* The same as total records above, but includes only the back versions. If the figure is 0 then garbage collection has taken place and removed the back versions - see above for an example.

*Max versions:* If a record has been updated many times, the `max versions` statistic shows you the number of back versions of the record (or records) in question. In a table where all the rows have been updated 7 times, but one has been updated 20 times, this statistic will report a value of 20. If the figure is 0.00 then garbage collection has taken place and removed the back versions - see above for an example.

[back to top of page](#)

## If you have database corruption

In the unlikely event of a [database corruption](#), your `gstat` output may have the following within the report:

```
Database file sequence:
File /opt/firebird/examples/empbuild/corrupt.fdb is the only file

Analyzing database pages ...
Expected b-tree bucket on page 337334 from 146314
```

If you do ever see a message like the above, displayed just after the header information, you are advised to immediately shut down all connections to the database, make an operating system level copy of the database file(s) and attempt to run `gbak` against the database to take a full backup. Using `nbackup` may copy the database happily, but not report any errors. `gbak`, on the other hand, will flag up errors.

From:  
<http://ibexpert.com/docu/> - IBExpert

Permanent link:  
<http://ibexpert.com/docu/doku.php?id=01-documentation:01-10-firebird-command-line-utilities:firebird-database-statistics:gstat-examples-and-interpretation>

Last update: 2023/08/07 14:59

