VACUUM Oct 25, 2012

When records are deleted from a table through an SQL DELETE, the deleted records are referred to as "dead tuples". They are not removed from the db and the space they reside in is NOT MARKED AS BEING AVAILABLE FOR RE-USE. Dead tuples are also created in the case of updated records and records involved in a transaction rollback. The VACUUM command must be run to make the "dead tuple" space available for reuse.

Running the vacuum command (without the "full" option) marks previously deleted (or updated) records as being available for re-use within the table. It does not lock the table being vacuumed. Running vacuum with the "analyze" option reads the records in the tables and generates statistics used by the queries. This information is stored in the pg_statistics table. Changes to the database made by a vacuum run are written to the transaction logs (WALs).

The vacuum command cannot be executed from within a function or a transaction.

VACUUM FULL

A "vacuum full" attempts to remove deleted or updated records from the tables to make the space reusable by other tables. It physically reorders the tables. While "vacuum full" is running, an exclusive lock is placed on the table being vacuumed. This locks the table for both reads and writes. A "vacuum full" run is NOT necessary to be run on a regular basis.

Before 9.0, VACUUM FULL required a REINDEX afterwards if you want to keep decent performance.

With 9.0, it is no longer required because the new VACUUM FULL doesn't bloat the index anymore.

Table and index bloat is a big problem in postgres databases. These days, PostgreSQL9.0 is widely used for production use and it's vital to remind people about changes in most important bloat removal tool called "VACUUM FULL".

Until PostgreSQL 9.0, VACUUM FULL was time consuming and DBA always stayed away from it and used CLUSTER instead.

The VACUUM FULL statement recovers free space from a table to reduce its size from bloated tables, mostly when VACUUM itself hasn't been run frequently enough. Before PostgreSQL 9.0 , it was time consuming and slow because of the way it was executed. Records were read and moved one by one from their source block to a block closer to the beginning of the table. Once the end of the table was emptied, this empty part was removed. This method was very inefficient. Moving records one by one creates a lot of random IO. Additionally, during this reorganization, indexes had to be maintained, making everything even more costly, and fragmenting indexes. It was therefore advised to reindex a table just after a VACUUM FULL.

In PostgreSQL 9.0, the VACUUM FULL statement creates a new table from the current one, copying all the records sequentially. Once all records are copied, indexes are created back, and the old table is destroyed and replaced. This has the advantage of being much faster. VACUUM

FULL still needs an **EXCLUSIVE LOCK** during the entire operation. The only drawback of this method compared to the old one, is that the new VACUUM FULL uses as much as two times the size of the table and indexes on disk, as it is creating a new versions of it.
VACCUM FULL on PostgreSQL 9.0 is way faster than previous versions. Moreover, VACUUM FULL has a couple of advantages over CLUSTER

1) it's faster than CLUSTER because it doesn't have to build new table using ORDER by clause
2) you can run VACUUM FULL on tables on which there isn't any index.

VACUUM ALL
The postgres documentation recommends that sites routinely run a "vacuum all" to be certain that all databases (including template1) are vacuumed. This can be done using the following command:

```
vacuumdb  -a  -U postgres
```

Note that the "-U postgres" must be specified on the command line for this to work correctly. Running "vacuumdb -a" without "-U postgres" while logged in as user=postgres does NOT work.

"vacuum" and "vacuum full" can be run for an entire database or for individual tables. See Section 21.1.1 of the PostgreSQL 7.4 Documentation entitled "Recovering disk space" for information on strategies for running VACUUM.

Auto Vacuum
Version 8.2.x introduced the concept of auto vacuum where a vacuum job will be automatically run if certain conditions are met. Also, only tables which need to be vacuumed (based on a set of configurable parameters) are vacuumed. By default, this feature is OFF in Version 8.2. In Versions 8.3.x and 8.4.x, this feature is ON by default.
Keep autovacuum enabled, and if required make it run *more* not less. If autovacuum is keeping up with your database's write load on 8.4 and above, you should not need to vacuum manually. However, there may be cases where a manual vacuum is necessary.

Q: What are the general guidelines under which autovacuum will trigger? I was unaware it was turned on by default for the newer versions. Would it be worthwhile to leave the manual vacuuming on? Currently it runs immediately after large sections of the tables are deleted. Or would it be expected that autovac would pick these changes up and run anyway?

A: Until 8.3, autovacuum was more of a proof of concept rather than production ready code. By 8.3 two things had happened, vacuum costing, which is important so you can tune vacuuming / autovacuuming to your hardware and usage patterns, and multi-threaded autovacuuming daemon, which meant that autovac could now handle the scenario where one or more table would take a long (sometimes very long) time to vacuum, especially with costing factors slowing it down, and another table would get bloated while waiting its turn. With a server with LOTS of random IO capability you can run quite a few threads at once, since each one is only a small impact against the maximum IO of the drive array. If you've got 1,000 tables and a couple dozen big ones that can take 30 minutes or more to vacuum, it's a good thing to be able to run autovac on more than one at a time.

The next HUGE improvement came with 8.4, which took the free space map and put it on the drives, removing the need to constantly monitor and adjust free space map to prevent blowout. If you've got a well tuned <= pg 8.3 you're ok. If you need to tune an older version, it's often easier AND safer to migrate to 8.4 or above.

Q: Why does autovacuum ignore some of the tables?

A: The table may have not had enough updates or deletes to trigger a vacuum. Are these insert-only tables? When you look at pg_stat_user_tables, check the n_tup_upd and n_tup_del columns.

```
IF
autovacuum_vacuum_threshold +
(autovacuum_vacuum_scale_factor * rows in the table) < n_dead_tup
```

in pg_stat_user_tables, then the table should be autovacuum'd. If it hasn't yet reached this number, it won't yet be a candidate.

Explanation of Vacuum Log Output

The last few lines of output from running a vacuum on an OHD db look as follows:

```
INFO: free space map: 901 relations, 6879 pages stored; 74608 total
pages needed
DETAIL: allocated FSM size: 1000 relations + 20000 pages = 178 kB
shared memory.
VACUUM
```

```
On the "INFO" line, "901 relations" signifies that a total of 901
tables currently exist across all databases on the server.

In the "DETAIL" line, FSM is an acronym for Free Space Map.  This line
shows that space has been allocated for a maximum of 1000 tables and
20000 pages for all postgresql databases on the server.
```

Vacuum of template1 Database
The reason for vacuuming template1 is that vacuum also resets the transaction ID number and prevents what is called "transaction ID wraparound". From what I can understand of this occurrence, all postgres db transactions are assigned an ID (a number). If the transaction ID number gets too big, it will "wraparound" to some smaller number possibly overwriting old transaction information with a subsequent loss of data. This can be expected to occur at sites (such as NWRFC) with huge numbers of daily database transactions. Note: **This occurrence has nothing to do with filling up disk space so the monitors on disk space will not catch this problem.**

postgres generates warning messages in the postgres log when transaction ID wraparound is imminent. These messages were seen at NWRFC. We have also seen them here at OHD. Until the vacuum of template1 is occurring regularly, the only way to watch for transaction ID wraparound is to monitor the postgres logs.

Beginning in Version 8.2, the transaction ID number is stored for each table instead of for each database.

Transaction ID
```
DELETE simply marks the tuple in the page as deleted (not visible) to
transactions with transaction ids larger than the DELETE transaction's
id (assuming it commits). So, once all transactions with transaction
ids lower than that DELETE transaction's id commit, the tuple can be
considered dead since nothing can see it anymore. VACUUM looks for
these dead tuples and adds them to the FSM. INSERTs and UPDATEs (and
COPYs) then look to the free space map for a dead tuple first when
space is needed for a new tuple before allocating space in hopes of
avoiding that space allocation by reusing the dead tuple's space.
```

Running vacuum at AWIPS Sites
The SwEG's policy on running vacuum is to have the postgres cron submit vacuum runs for each of the standard AWIPS databases. This will prevent the possibility of multiple vacuum runs executing at the same time which can cause a slowdown in the server. A vacuum/analyze of the IHFS db has been scheduled to be submitted from the postgres cron every 4 hours.

Vacuum Script
The following script is submitted via the postgres cron on dx1 at all AWIPS sites to vacuum the PostgreSQL databases. The filename is /awips/ops/bin/vacuum_pgdb.

```
#!/bin/bash
#
# NAME
#   vacuum_pgdb - Vacuum a postgres database
#
# SYNOPSIS
#   vacuum_db -d db_name [-z]
#
# DESCRIPTION
#   This script calls the vacuumdb executable to vacuum AWIPS
databases.
#   It will normally be run from the "postgres admin user" cron.
#
#   The command line of the script is
#
#       vacuum_db -d db_name,... [-z]
#
#       db_name = the name of the database to be vacuumed
#
#   or
#       vacuum_db -a -x db_name,... [-z]
#
#       db_name = the name of the database to exclude from the vacuum
#
#   The "-z" option is optional.  If it appears on the command line,
#   then the vacuum will also perform an "analyze".
#
#   This script logs output to /data/logs/fxa/vacuum_${DBNAME}_MMDD.
#
# HISTORY
#   4/08/2005   Original Version (Paul Tilles)
#   4/19/2005   Updates for environment vars, command line options
#
#######################################################################

USAGE="Usage: $0 -a|-d dbname_list [-x exclude_list] [-z]"
FXA_HOME=${FXA_HOME:-~fxa}
unset ANALYZE
VACUUM="vacuum"
VACUUM_ALL=0
unset DBNAME_ARRAY
unset EXCLUDE_ARRAY

# Read the command line args
while getopts :ad:x:z opt ; do
    case $opt in
        a ) VACUUM_ALL=1
            ;;
        d ) DBNAME_ARRAY=( ${OPTARG//,/ } )
            ;;
      x ) EXCLUDE_ARRAY=( ${OPTARG//,/ } )
          ;;
        z ) ANALYZE="--analyze"
            VACUUM="vacuum analyze"
            ;;
        * ) echo $USAGE;
            exit 1
            ;;
    esac
done
```

```
if [ -z "${DBNAME_ARRAY[*]}" -a $VACUUM_ALL -eq 0 ] ; then
    echo $USAGE
    exit 1
fi

# Source the AWIPS and PostgreSQL environments
. $FXA_HOME/readenv.sh
. postgresenv.sh

PSQL_BIN_DIR=$PG_INSTALL/bin

if [ $VACUUM_ALL -ne 0 ] ; then
    DBNAME_ARRAY=( $($PG_INSTALL/bin/psql -U postgres --list --tuples-
only \
      | while read _DBNAME _JUNK; do \
      if [ "$_DBNAME" != "template0" -a "$_DBNAME" != "template1" ] ;
then \
      echo $_DBNAME; fi; done) )
fi

for EXCLUDE in ${EXCLUDE_ARRAY[*]} ; do
    let "I = 0"
    while [ ! -z "${DBNAME_ARRAY[$I]}" ] ; do
      if [ "${DBNAME_ARRAY[$I]}" = "$EXCLUDE" ] ; then
          unset DBNAME_ARRAY[$I]
      fi
      let "I = $I + 1"
    done
done

if [ -z "${DBNAME_ARRAY[*]}" ] ; then
    echo "Nothing to vacuum!" > /dev/stderr
    exit 1
fi

########################################################################
# Run vacuumdb
# Write database name, begin time and end time to log

EXIT=0
for DBNAME in ${DBNAME_ARRAY[*]} ; do

    LOGFILE=$LOG_DIR/vacuum_${DBNAME}_$(date -u +%m%d_%H%M)

# We should not need to do this, all databases should be owned by
pguser
    USERNAME=$($PG_INSTALL/bin/psql -U postgres --list --tuples-only |
\
      while read _DBNAME _DELIM _USERNAME _JUNK ; do \
      if [ "$_DBNAME" = "$DBNAME" ] ; then echo $_USERNAME; fi; \
      done)

    echo $(date +"%b %d %T") BEGIN $VACUUM $DBNAME as $USERNAME >>
$LOGFILE
```

```
    $PSQL_BIN_DIR/vacuumdb -v $ANALYZE -U $USERNAME $DBNAME >> $LOGFILE
2>&1
    RETURN=$?

    if [ $RETURN -ne 0 ] ; then
      EXIT=$RETURN
    fi

    DTZ=`date -u +%T`
    echo $(date +"%b %d %T") END $VACUUM $DBNAME EXIT_CODE=$RETURN >>
$LOGFILE

done

exit $EXIT
```

A log file is generated by each vacuum run. Logs generated by the execution of the /awips/ops/bin/vacuum_pgdb script are written to the $LOG_DIR directory which normally points to the /data/logs/fxa directory. These log files will be monitored to watch for problems such as a slow increase in size of the db over time. We also hope to glean information from the logs which will be used to tweak the configuration parameters.

Submitting from cron
The vacuum_pgdb script is submitted via the cron on dx1. The cron file is located in

/etc/ha.d/cron.d/dx1cron

Other Info
The following site offers an interesting discussion concerning vacuuming:

http://pgsqld.active-venture.com/routine-vacuuming.html

Analyzing Vacuum Output
Q: I got the following output at the end of my vacuum :

```
INFO:  free space map:  260 relations, 20604 pages stored;  52512 total pages needed
DETAIL:  Allocated FSM size;  1000 relations + 20000 pages = 178 kB shared memory
```

This output is from a Version 7.4.x "vacuum all". How should this output be interpreted?

A: It appears your FSM is a bit too small. While it can track all of the relations you have, it's not able to store information about all of the pages that contain free space. As a result, there is probably a lot of fragmented data (spaces marked as free, but the tuples aren't being re-used because the FSM isn't tracking them). I'd increase the amount of free pages you are tracking with the FSM.

Q: I using Postgresql 8.1 and during vacuum at night time, I am getting the following log:

```
number of page slots needed (2520048) exceeds max_fsm_pages (356656)
```

Do I need to increase max_fsm_pages to 2520048? Does it have any bad affect?

A: You don't *have* to do it. The consequences of not doing it are:

1) your server will not know all of the pages in the files holding your database that there are empty slots available for use.
2) because of that lack of knowledge, it may then allocate new pages to hold your data, causing potentially more bloat, and the need for even more FSM pages.
3) Allocating new pages usually costs more than just filling in space on existing page, so your system slows down.

If I were you, I'd set the FSM pages to double what your current need is, run vacuum again, and you should be good for a while. It will unfortunately, require a restart of your postgres server.

Vacuum and Open Transactions
Q: If a java program connects to the DB and does "begin;" and then internally does a "sleep 6 days", does that cause any issues other than eating a connection to the database?

A: In recent versions of PG, no. Before about 8.3 it was a Really Bad Idea because the open transaction would prevent VACUUM from reclaiming storage.

It's *still* a Really Bad Idea to begin a transaction, do something, and then sleep 6 days. But "BEGIN" without any following commands has been fixed to be harmless, mainly because there are so many badly designed clients that do exactly that.

CHPS Database
On Nov 5, 2010, NWRFC reported a problem with vacuuming the CHPS db. Because of an event, the db had not been vacuumed for 3 days. When they attempted to do a "vacuum –full" on the db, it took many hours and did not finish. The vacuum was "stuck" on the time series table. This is a very large table with a TOAST table associated with it. The solution to the problem was to run CLUSTER on the primary key index of the table followed by a "vacuum full". The CLUSTER on the index took approx 1 hour. The "vacuum full" took approx 10 minutes.   CHPS is running V 8.2.x.
To run cluster,

```
psql db_name
cluster index_name on table_name;
```

The cluster command reorders the records in the table according to the specified index.

Vacuum of a Logging Table

If it's purely an insert-only table, such as a logging table, then in principle you only need periodic ANALYZEs and not any VACUUMs.

VACUUM could still be worthwhile though, because (a) it will set commit hint bits on all pages and (b) it will set visibility-map bits on all pages. An ANALYZE would only do those things for the random sample of pages that it visits. While neither of those things are critical, they do offload work from future queries that would otherwise have to do that work in-line. So if you've got a maintenance window where the database
isn't answering queries anyway, it could be worthwhile to run a VACUUM just to get those bits set.

Disable auto-vacuum on a Specific Table

Exactly how you do it depends on the server version. In versions earlier than 8.4 you have to manually insert a tuple in the pg_autovacuum catalog, with its "enabled" flag set to false and ensure that all other settings are -1 (not zero).

In 8.4 and up, just do

```
ALTER TABLE foo SET (autovacuum_enabled = false)
```