<https://www.dropbox.com/s/k1vauzxxhyh1fnb/SQL%20Server%202019%20Diagnostic%20Information%20Queries.sql?e=1&dl=0>

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579  580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598  599  600  601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623  624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639  640  641  642  643  644  645  646  647  648  649  650  651  652  653  654  655  656  657  658  659  660  661  662  663  664  665  666  667  668  669  670  671  672  673  674  675  676  677  678  679  680  681  682  683  684  685  686  687  688  689  690  691  692  693  694  695  696  697  698  699  700  701  702  703  704  705  706  707  708  709  710  711  712  713  714  715  716  717  718  719  720  721  722  723  724  725  726  727  728  729  730  731  732  733  734  735  736  737  738  739  740  741  742  743  744  745  746  747  748  749  750  751  752  753  754  755  756  757  758  759  760  761  762  763  764  765  766  767  768  769  770  771  772  773  774  775  776  777  778  779  780  781  782  783  784  785  786  787  788  789  790  791  792  793  794  795  796  797  798  799  800  801  802  803  804  805  806  807  808  809  810  811  812  813  814  815  816  817  818  819  820  821  822  823  824  825  826  827  828  829  830  831  832  833  834  835  836  837  838  839  840  841  842  843  844  845  846  847  848  849  850  851  852  853  854  855  856  857  858  859  860  861  862  863  864  865  866  867  868  869  870  871  872  873  874  875  876  877  878  879  880  881  882  883  884  885  886  887  888  889  890  891  892  893  894  895  896  897  898  899  900  901  902  903  904  905  906  907  908  909  910  911  912  913  914  915  916  917  918  919  920  921  922  923  924  925  926  927  928  929  930  931  932  933  934  935  936  937  938  939  940  941  942  943  944  945  946  947  948  949  950  951  952  953  954  955  956  957  958  959  960  961  962  963  964  965  966  967  968  969  970  971  972  973  974  975  976  977  978  979  980  981  982  983  984  985  986  987  988  989  990  991  992  993  994  995  996  997  998  999  1000  1001  1002  1003  1004  1005  1006  1007  1008  1009  1010  1011  1012  1013  1014  1015  1016  1017  1018  1019  1020  1021  1022  1023  1024  1025  1026  1027  1028  1029  1030  1031  1032  1033  1034  1035  1036  1037  1038  1039  1040  1041  1042  1043  1044  1045  1046  1047  1048  1049  1050  1051  1052  1053  1054  1055  1056  1057  1058  1059  1060  1061  1062  1063  1064  1065  1066  1067  1068  1069  1070  1071  1072  1073  1074  1075  1076  1077  1078  1079  1080  1081  1082  1083  1084  1085  1086  1087  1088  1089  1090  1091  1092  1093  1094  1095  1096  1097  1098  1099  1100  1101  1102  1103  1104  1105  1106  1107  1108  1109  1110  1111  1112  1113  1114  1115  1116  1117  1118  1119  1120  1121  1122  1123  1124  1125  1126  1127  1128  1129  1130  1131  1132  1133  1134  1135  1136  1137  1138  1139  1140  1141  1142  1143  1144  1145  1146  1147  1148  1149  1150  1151  1152  1153  1154  1155  1156  1157  1158  1159  1160  1161  1162  1163  1164  1165  1166  1167  1168  1169  1170  1171  1172  1173  1174  1175  1176  1177  1178  1179  1180  1181  1182  1183  1184  1185  1186  1187  1188  1189  1190  1191  1192  1193  1194  1195  1196  1197  1198  1199  1200  1201  1202  1203  1204  1205  1206  1207  1208  1209  1210  1211  1212  1213  1214  1215  1216  1217  1218  1219  1220  1221  1222  1223  1224  1225  1226  1227  1228  1229  1230  1231  1232  1233  1234  1235  1236  1237  1238  1239  1240  1241  1242  1243  1244  1245  1246  1247  1248  1249  1250  1251  1252  1253  1254  1255  1256  1257  1258  1259  1260  1261  1262  1263  1264  1265  1266  1267  1268  1269  1270  1271  1272  1273  1274  1275  1276  1277  1278  1279  1280  1281  1282  1283  1284  1285  1286  1287  1288  1289  1290  1291  1292  1293  1294  1295  1296  1297  1298  1299  1300  1301  1302  1303  1304  1305  1306  1307  1308  1309  1310  1311  1312  1313  1314  1315  1316  1317  1318  1319  1320  1321  1322  1323  1324  1325  1326  1327  1328  1329  1330  1331  1332  1333  1334  1335  1336  1337  1338  1339  1340  1341  1342  1343  1344  1345  1346  1347  1348  1349  1350  1351  1352  1353  1354  1355  1356  1357  1358  1359  1360  1361  1362  1363  1364  1365  1366  1367  1368  1369  1370  1371  1372  1373  1374  1375  1376  1377  1378  1379  1380  1381  1382  1383  1384  1385  1386  1387  1388  1389  1390  1391  1392  1393  1394  1395  1396  1397  1398  1399  1400  1401  1402  1403  1404  1405  1406  1407  1408  1409  1410  1411  1412  1413  1414  1415  1416  1417  1418  1419  1420  1421  1422  1423  1424  1425  1426  1427  1428  1429  1430  1431  1432  1433  1434  1435  1436  1437  1438  1439  1440  1441  1442  1443  1444  1445  1446  1447  1448  1449  1450  1451  1452  1453  1454  1455  1456  1457  1458  1459  1460  1461  1462  1463  1464  1465  1466  1467  1468  1469  1470  1471  1472  1473  1474  1475  1476  1477  1478  1479  1480  1481  1482  1483  1484  1485  1486  1487  1488  1489  1490  1491  1492  1493  1494  1495  1496  1497  1498  1499  1500  1501  1502  1503  1504  1505  1506  1507  1508  1509  1510  1511  1512  1513  1514  1515  1516  1517  1518  1519  1520  1521  1522  1523  1524  1525  1526  1527  1528  1529  1530  1531  1532  1533  1534  1535  1536  1537  1538  1539  1540  1541  1542  1543  1544  1545  1546  1547  1548  1549  1550  1551  1552  1553  1554  1555  1556  1557  1558  1559  1560  1561  1562  1563  1564  1565  1566  1567  1568  1569  1570  1571  1572  1573  1574  1575  1576  1577  1578  1579  1580  1581  1582  1583  1584  1585  1586  1587  1588  1589  1590  1591  1592  1593  1594  1595  1596  1597  1598  1599  1600  1601  1602  1603  1604  1605  1606  1607  1608  1609  1610  1611  1612  1613  1614  1615  1616  1617  1618  1619  1620  1621  1622  1623  1624  1625  1626  1627  1628  1629  1630  1631  1632  1633  1634  1635  1636  1637  1638  1639  1640  1641  1642  1643  1644  1645  1646  1647  1648  1649  1650  1651  1652  1653  1654  1655  1656  1657  1658  1659  1660  1661  1662  1663  1664  1665  1666  1667  1668  1669  1670  1671  1672  1673  1674  1675  1676  1677  1678  1679  1680  1681  1682  1683  1684  1685  1686  1687  1688  1689  1690  1691  1692  1693  1694  1695  1696  1697  1698  1699  1700  1701  1702  1703  1704  1705  1706  1707  1708  1709  1710  1711  1712  1713  1714  1715  1716  1717  1718  1719  1720  1721  1722  1723  1724  1725  1726  1727  1728  1729  1730  1731  1732  1733  1734  1735  1736  1737  1738  1739  1740  1741  1742  1743  1744  1745  1746  1747  1748  1749  1750  1751  1752  1753  1754  1755  1756  1757  1758  1759  1760  1761  1762  1763  1764  1765  1766  1767  1768  1769  1770  1771  1772  1773  1774  1775  1776  1777  1778  1779  1780  1781  1782  1783  1784  1785  1786  1787  1788  1789  1790  1791  1792  1793  1794  1795  1796  1797  1798  1799  1800  1801  1802  1803  1804  1805  1806  1807  1808  1809  1810  1811  1812  1813  1814  1815  1816  1817  1818  1819  1820  1821  1822  1823  1824  1825  1826  1827  1828  1829  1830  1831  1832  1833  1834  1835  1836  1837  1838  1839  1840  1841  1842  1843  1844  1845  1846  1847  1848  1849  1850  1851  1852  1853  1854  1855  1856  1857  1858  1859  1860  1861  1862  1863  1864  1865  1866  1867  1868  1869  1870  1871  1872  1873  1874  1875  1876  1877  1878  1879  1880  1881  1882  1883  1884  1885  1886  1887  1888  1889  1890  1891  1892  1893  1894  1895  1896  1897  1898  1899  1900  1901  1902  1903  1904  1905  1906  1907  1908  1909  1910  1911  1912  1913  1914  1915  1916  1917  1918  1919  1920  1921  1922  1923  1924  1925  1926  1927  1928  1929  1930  1931  1932  1933  1934  1935  1936  1937  1938  1939  1940  1941  1942  1943  1944  1945  1946  1947  1948  1949  1950  1951  1952  1953  1954  1955  1956  1957  1958  1959  1960  1961  1962  1963  1964  1965  1966  1967  1968  1969  1970  1971  1972  1973  1974  1975  1976  1977  1978  1979  1980  1981  1982  1983  1984  1985  1986  1987  1988  1989  1990  1991  1992  1993  1994  1995  1996  1997  1998  1999  2000  2001  2002  2003  2004  2005  2006  2007  2008  2009  2010  2011  2012  2013  2014  2015  2016  2017  2018  2019  2020  2021  2022  2023  2024  2025  2026  2027  2028  2029  2030  2031  2032  2033  2034  2035  2036  2037  2038  2039  2040  2041  2042  2043  2044  2045  2046  2047  2048  2049  2050  2051  2052  2053  2054  2055  2056  2057  2058  2059  2060  2061  2062  2063  2064  2065  2066  2067  2068  2069  2070  2071  2072  2073  2074  2075  2076  2077  2078  2079  2080  2081  2082  2083  2084  2085  2086  2087  2088  2089  2090  2091  2092  2093  2094  2095  2096  2097  2098  2099  2100  2101  2102  2103  2104  2105  2106  2107  2108  2109  2110  2111  2112  2113  2114  2115  2116  2117  2118  2119  2120  2121  2122  2123  2124  2125  2126  2127  2128  2129  2130  2131  2132  2133  2134  2135  2136  2137  2138  2139  2140  2141  2142  2143  2144  2145  2146  2147  2148  2149  2150  2151  2152  2153  2154  2155  2156  2157  2158  2159  2160  2161  2162  2163  2164  2165  2166  2167  2168  2169  2170  2171  2172  2173  2174  2175  2176  2177  2178  2179  2180  2181  2182  2183  2184  2185  2186  2187  2188  2189  2190  2191  2192  2193  2194  2195  2196  2197  2198  2199  2200  2201  2202  2203  2204  2205  2206  2207  2208  2209  2210  2211  2212  2213  2214  2215  2216  2217  2218  2219  2220  2221  2222  2223  2224  2225  2226  2227  2228  2229  2230  2231  2232  2233  2234  2235  2236  2237  2238  2239  2240  2241  2242 | *-- SQL Server 2019 Diagnostic Information Queries*  *-- Glenn Berry*  *-- Last Modified: June 3, 2024*  *-- https://glennsqlperformance.com/*  *-- https://sqlserverperformance.wordpress.com/*  *-- YouTube: https://bit.ly/2PkoAM1*  *-- Twitter: GlennAlanBerry*  *-- Diagnostic Queries are available here*  *-- https://glennsqlperformance.com/resources/*  *-- YouTube video demonstrating these queries*  *-- https://bit.ly/3aXNDzJ*  *-- Please make sure you are using the correct version of these diagnostic queries for your version of SQL Server*  *-- If you like PowerShell, there is a very useful community solution for running these queries in an automated fashion*  *-- https://dbatools.io/*  *-- Invoke-DbaDiagnosticQuery*  *-- https://docs.dbatools.io/Invoke-DbaDiagnosticQuery*  *--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *--\* Copyright (C) 2024 Glenn Berry*  *--\* All rights reserved.*  *--\**  *--\**  *--\* You may alter this code for your own \*non-commercial\* purposes. You may*  *--\* republish altered code as long as you include this copyright and give due credit.*  *--\**  *--\**  *--\* THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF*  *--\* ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED*  *--\* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A*  *--\* PARTICULAR PURPOSE.*  *--\**  *--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *-- Check the major product version to see if it is SQL Server 2019 CTP 2 or greater*  **IF** **NOT** **EXISTS** (**SELECT** \* **WHERE** CONVERT(**varchar**(128), SERVERPROPERTY('ProductVersion')) **LIKE** '15%')  **BEGIN**  **DECLARE** @ProductVersion **varchar**(128) = CONVERT(**varchar**(128), SERVERPROPERTY('ProductVersion'));  **RAISERROR** ('Script does not match the ProductVersion [%s] of this instance. Many of these queries may not work on this version.' , 18 , 16 , @ProductVersion);  **END**  **ELSE**  **PRINT** N'You have the correct major version of SQL Server for this diagnostic information script';    *-- Instance level queries \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *-- SQL and OS Version information for current instance (Query 1) (Version Info)*  **SELECT** @@SERVERNAME **AS** [Server Name], @@VERSION **AS** [SQL Server and OS Version Info];  *------*  *-- SQL Server 2019 Builds*  *-- Build Description Release Date URL to KB Article*  *-- 15.0.1000.34 CTP 2.0 9/24/2018*  *-- 15.0.1100.94 CTP 2.1 11/7/2018*  *-- 15.0.1200.24 CTP 2.2 12/6/2018*  *-- 15.0.1300.359 CTP 2.3 3/1/2019*  *-- 15.0.1400.75 CTP 2.4 3/26/2019*  *-- 15.0.1500.28 CTP 2.5 4/23/2019*  *-- 15.0.1600.8 CTP 3.0 5/22/2019*  *-- 15.0.1700.37 CTP 3.1 6/26/2019*  *-- 15.0.1800.32 CTP 3.2 7/24/2019*  *-- 15.0.1900.25 RC1/RC1 Refresh 8/29/2019*  *-- 15.0.2000.5 RTM 11/4/2019*  *-- 15.0.2070.41 GDR1 11/4/2019 https://support.microsoft.com/en-us/help/4517790/servicing-update-for-sql-server-2019-rtm*  *-- 15.0.4003.23 CU1 1/7/2020 https://support.microsoft.com/en-us/help/4527376/cumulative-update-1-for-sql-server-2019*  *-- 15.0.4013.40 CU2 2/13/2020 https://support.microsoft.com/en-us/help/4536075/cumulative-update-2-for-sql-server-2019*  *-- 15.0.4023.6 CU3 3/12/2020 https://support.microsoft.com/en-us/help/4538853/cumulative-update-3-for-sql-server-2019*  *-- 15.0.4033.1 CU4 3/31/2020 https://support.microsoft.com/en-us/help/4548597/cumulative-update-4-for-sql-server-2019*  *-- 15.0.4043.16 CU5 6/22/2020 https://support.microsoft.com/en-us/help/4552255/cumulative-update-5-for-sql-server-2019*  *-- 15.0.4053.23 CU6 8/4/2020 https://support.microsoft.com/en-us/help/4563110/cumulative-update-6-for-sql-server-2019*  *-- 15.0.4063.15 CU7 9/2/2020 -- CU7 was removed by Microsoft*  *-- 15.0.4073.23 CU8 10/1/2020 https://support.microsoft.com/en-in/help/4577194/cumulative-update-8-for-sql-server-2019*  *-- 15.0.4083.2 CU8 Security Update 1/12/2021 https://support.microsoft.com/en-us/help/4583459/kb4583459-security-update-for-sql-server-2019-cu8*  *-- 15.0.4102.2 CU9 2/11/2021 https://support.microsoft.com/en-in/help/5000642/cumulative-update-9-for-sql-server-2019*  *-- 15.0.4123.1 CU10 4/6/2021 https://support.microsoft.com/en-us/topic/kb5001090-cumulative-update-10-for-sql-server-2019-b6b696ec-6598-48d9-80ee-f1b85d7a508b*  *-- 15.0.4138.2 CU11 6/10/2021 https://support.microsoft.com/en-us/topic/kb5003249-cumulative-update-11-for-sql-server-2019-657b2977-a0f1-4e1f-8b93-8c2ca8b6bef5*  *-- 15.0.4153.1 CU12 8/4/2021 https://support.microsoft.com/en-us/topic/kb5004524-cumulative-update-12-for-sql-server-2019-45b2d82a-c7d0-4eb8-aa17-d4bad4059987*  *-- 15.0.4178.1 CU13 10/5/2021 https://support.microsoft.com/en-us/topic/kb5005679-cumulative-update-13-for-sql-server-2019-5c1be850-460a-4be4-a569-fe11f0adc535*  *-- 15.0.4188.2 CU14 11/22/2021 https://support.microsoft.com/sl-si/topic/kb5007182-cumulative-update-14-for-sql-server-2019-67b00a61-4f30-4a36-a5db-b506c47e563b*  *-- 15.0.4198.2 CU15 1/27/2022 https://support.microsoft.com/en-us/topic/kb5008996-cumulative-update-15-for-sql-server-2019-4b6a8ee9-1c61-482d-914f-36e429901fb6*  *-- 15.0.4223.1 CU16 4/18/2022 https://support.microsoft.com/en-us/topic/kb5011644-cumulative-update-16-for-sql-server-2019-74377be1-4340-4445-93a7-ff843d346896*  *-- 15.0.4236.7 CU16 Security Update 6/14/2022 https://support.microsoft.com/en-us/topic/kb5014353-description-of-the-security-update-for-sql-server-2019-cu16-june-14-2022-f0afe659-bd19-4c87-a417-a4c67a47e644*  *-- 15.0.4249.2 CU17 8/11/2022 https://support.microsoft.com/en-us/topic/kb5016394-cumulative-update-17-for-sql-server-2019-3033f654-b09d-41aa-8e49-e9d0c353c5f7*  *-- 15.0.4261.1 CU18 9/28/2022 https://support.microsoft.com/en-us/topic/kb5017593-cumulative-update-18-for-sql-server-2019-5fa00c36-edeb-446c-94e3-c4882b7526bc*  *-- 15.0.4280.7 CU18 GDR 2/14/2023 https://support.microsoft.com/en-us/topic/kb5021124-description-of-the-security-update-for-sql-server-2019-cu18-february-14-2023-cfb75a0a-33dc-4e05-8645-4cf16fcec049*  *-- 15.0.4298.1 CU19 2/16/2023 https://support.microsoft.com/en-us/topic/kb5023049-cumulative-update-19-for-sql-server-2019-b63d7163-e2e7-46f7-b50a-c3d1f2913219*  *-- 15.0.4312.2 CU20 4/13/2023 https://support.microsoft.com/en-us/topic/kb5024276-cumulative-update-20-for-sql-server-2019-4b282be9-b559-46ac-9b6a-badbd44785d2*  *-- 15.0.4316.3 CU21 6/15/2022 https://learn.microsoft.com/en-us/troubleshoot/sql/releases/sqlserver-2019/cumulativeupdate21*  *-- 15.0.4322.2 CU22 8/14/2023 https://learn.microsoft.com/en-us/troubleshoot/sql/releases/sqlserver-2019/cumulativeupdate22*  *-- 15.0.4326.1 CU22 + GDR 10/10/2023 https://support.microsoft.com/en-us/topic/kb5029378-description-of-the-security-update-for-sql-server-2019-cu22-october-10-2023-f4b5c5fb-b4cd-4599-8e5b-2a54dab85a33*  *-- 15.0.4335.1 CU23 10/12/2023 https://learn.microsoft.com/en-us/troubleshoot/sql/releases/sqlserver-2019/cumulativeupdate23*  *-- 15.0.4345.5 CU24 12/14/2023 https://learn.microsoft.com/en-us/troubleshoot/sql/releases/sqlserver-2019/cumulativeupdate24*  *-- 15.0.4355.3 CU25 2/15/2024 https://learn.microsoft.com/en-us/troubleshoot/sql/releases/sqlserver-2019/cumulativeupdate25*  *-- 15.0.4360.2 CU25 + GDR 4/9/2024 https://support.microsoft.com/en-us/topic/kb5036335-description-of-the-security-update-for-sql-server-2019-cu25-april-9-2024-eb3571d0-62ee-445e-9681-5715caf9bbc2*  *-- 15.0.4365.2 CU26 4/11/2024 https://learn.microsoft.com/en-us/troubleshoot/sql/releases/sqlserver-2019/cumulativeupdate26*  *-- How to determine the version, edition and update level of SQL Server and its components*  *-- https://bit.ly/2oAjKgW*  *-- SQL Server 2019 build versions*  *-- https://bit.ly/3EzGQZV*  *-- Performance and Stability Fixes in SQL Server 2019 CU Builds*  *-- https://bit.ly/3712NQQ*  *-- What's New in SQL Server 2019 (Database Engine)*  *-- https://bit.ly/2Q29fhz*  *-- What's New in SQL Server 2019*  *-- https://bit.ly/2PY442b*  *-- Announcing the Modern Servicing Model for SQL Server*  *-- https://bit.ly/2KtJ8SS*  *-- Update Center for Microsoft SQL Server*  *-- https://bit.ly/2pZptuQ*  *-- Download SQL Server Management Studio (SSMS)*  *-- https://bit.ly/1OcupT9*  *-- Download and install Azure Data Studio*  *-- https://bit.ly/2vgke1A*  *-- SQL Server 2019 Configuration Manager is SQLServerManager15.msc*  *-- SQL Server troubleshooting (Microsoft documentation resources)*  *-- http://bit.ly/2YY0pb1*  *-- Get socket, physical core and logical core count from the SQL Server Error log. (Query 2) (Core Counts)*  *-- This query might take a few seconds depending on the size of your error log*  **EXEC** sys.xp\_readerrorlog 0, 1, N'detected', N'socket';  *------*  *-- This can help you determine the exact core counts used by SQL Server and whether HT is enabled or not*  *-- It can also help you confirm your SQL Server licensing model*  *-- Be on the lookout for this message "using 40 logical processors based on SQL Server licensing"*  *-- (when you have more than 40 logical cores) which means grandfathered Server/CAL licensing*  *-- This query will return no results if your error log has been recycled since the instance was last started*  *-- Get selected server properties (Query 3) (Server Properties)*  **SELECT** SERVERPROPERTY('MachineName') **AS** [MachineName],  SERVERPROPERTY('ServerName') **AS** [ServerName],  SERVERPROPERTY('InstanceName') **AS** [Instance],  SERVERPROPERTY('IsClustered') **AS** [IsClustered],  SERVERPROPERTY('ComputerNamePhysicalNetBIOS') **AS** [ComputerNamePhysicalNetBIOS],  SERVERPROPERTY('Edition') **AS** [Edition],  SERVERPROPERTY('ProductLevel') **AS** [ProductLevel], *-- What servicing branch (RTM/SP/CU)*  SERVERPROPERTY('ProductUpdateLevel') **AS** [ProductUpdateLevel], *-- Within a servicing branch, what CU# is applied*  SERVERPROPERTY('ProductVersion') **AS** [ProductVersion],  SERVERPROPERTY('ProductMajorVersion') **AS** [ProductMajorVersion],  SERVERPROPERTY('ProductMinorVersion') **AS** [ProductMinorVersion],  SERVERPROPERTY('ProductBuild') **AS** [ProductBuild],  SERVERPROPERTY('ProductBuildType') **AS** [ProductBuildType], *-- Is this a GDR or OD hotfix (NULL if on a CU build)*  SERVERPROPERTY('ProductUpdateReference') **AS** [ProductUpdateReference], *-- KB article number that is applicable for this build*  SERVERPROPERTY('ProcessID') **AS** [ProcessID],  SERVERPROPERTY('Collation') **AS** [Collation],  SERVERPROPERTY('IsFullTextInstalled') **AS** [IsFullTextInstalled],  SERVERPROPERTY('IsIntegratedSecurityOnly') **AS** [IsIntegratedSecurityOnly],  SERVERPROPERTY('FilestreamConfiguredLevel') **AS** [FilestreamConfiguredLevel],  SERVERPROPERTY('IsHadrEnabled') **AS** [IsHadrEnabled],  SERVERPROPERTY('HadrManagerStatus') **AS** [HadrManagerStatus],  SERVERPROPERTY('InstanceDefaultDataPath') **AS** [InstanceDefaultDataPath],  SERVERPROPERTY('InstanceDefaultLogPath') **AS** [InstanceDefaultLogPath],  SERVERPROPERTY('InstanceDefaultBackupPath') **AS** [InstanceDefaultBackupPath],  SERVERPROPERTY('ErrorLogFileName') **AS** [ErrorLogFileName],  SERVERPROPERTY('BuildClrVersion') **AS** [Build CLR Version],  SERVERPROPERTY('IsXTPSupported') **AS** [IsXTPSupported],  SERVERPROPERTY('IsPolybaseInstalled') **AS** [IsPolybaseInstalled],  SERVERPROPERTY('IsAdvancedAnalyticsInstalled') **AS** [IsRServicesInstalled],  SERVERPROPERTY('IsTempdbMetadataMemoryOptimized') **AS** [IsTempdbMetadataMemoryOptimized];  *------*  *-- This gives you a lot of useful information about your instance of SQL Server,*  *-- such as the ProcessID for SQL Server and your collation*  *-- Note: Some columns will be NULL on older SQL Server builds*  *-- SERVERPROPERTY('IsTempdbMetadataMemoryOptimized') is a new option for SQL Server 2019*  *-- SERVERPROPERTY (Transact-SQL)*  *-- https://bit.ly/2eeaXeI*  *-- Get instance-level configuration values for instance (Query 4) (Configuration Values)*  **SELECT** name, **value**, value\_in\_use, minimum, maximum, [description], is\_dynamic, is\_advanced  **FROM** sys.configurations **WITH** (NOLOCK)  **ORDER** **BY** name **OPTION** (RECOMPILE);  *------*  *-- Focus on these settings:*  *-- automatic soft-NUMA disabled (should be 0 in most cases)*  *-- backup checksum default (should be 1)*  *-- backup compression default (should be 1 in most cases)*  *-- clr enabled (only enable if it is needed)*  *-- cost threshold for parallelism (depends on your workload)*  *-- lightweight pooling (should be zero)*  *-- max degree of parallelism (depends on your workload and hardware)*  *-- max server memory (MB) (set to an appropriate value, not the default)*  *-- optimize for ad hoc workloads (should be 1)*  *-- priority boost (should be zero)*  *-- remote admin connections (should be 1)*  *-- tempdb metadata memory-optimized (0 by default, some workloads may benefit by enabling)*  *-- sys.configurations (Transact-SQL)*  *-- https://bit.ly/2HsyDZI*  *-- Returns a list of all global trace flags that are enabled (Query 5) (Global Trace Flags)*  **DBCC** TRACESTATUS (-1);  *------*  *-- If no global trace flags are enabled, no results will be returned.*  *-- It is very useful to know what global trace flags are currently enabled as part of the diagnostic process.*  *-- Common trace flags that should be enabled in most cases*  *-- TF 3226 - Suppresses logging of successful database backup messages to the SQL Server Error Log*  *-- https://bit.ly/38zDNAK*  *-- TF 6534 - Enables use of native code to improve performance with spatial data. This is a startup trace flag only*  *-- https://bit.ly/2HrQUpU*  *-- TF 7745 - Prevents Query Store data from being written to disk in case of a failover or shutdown command*  *-- https://bit.ly/2GU69Km*  *-- TF 8121 - Fixes a system-wide low memory issue that occurs when SQL Server commits memory above the maximum server memory under the memory model with the Lock Pages In Memory (added in CU15)*  *-- https://learn.microsoft.com/en-US/troubleshoot/sql/releases/sqlserver-2019/cumulativeupdate15#bkmk\_14421838*  *-- DBCC TRACEON - Trace Flags (Transact-SQL)*  *-- https://bit.ly/2FuSvPg*  *-- SQL Server Process Address space info (Query 6) (Process Memory)*  *-- (shows whether locked pages is enabled, among other things)*  **SELECT** physical\_memory\_in\_use\_kb/1024 **AS** [SQL Server Memory Usage (MB)],  locked\_page\_allocations\_kb/1024 **AS** [SQL Server Locked Pages Allocation (MB)],  large\_page\_allocations\_kb/1024 **AS** [SQL Server Large Pages Allocation (MB)],  page\_fault\_count, memory\_utilization\_percentage, available\_commit\_limit\_kb,  process\_physical\_memory\_low, process\_virtual\_memory\_low  **FROM** sys.dm\_os\_process\_memory **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- You want to see 0 for process\_physical\_memory\_low*  *-- You want to see 0 for process\_virtual\_memory\_low*  *-- This indicates that you are not under internal memory pressure*  *-- If locked\_page\_allocations\_kb > 0, then LPIM is enabled*  *-- sys.dm\_os\_process\_memory (Transact-SQL)*  *-- https://bit.ly/3iUgQgC*  *-- How to enable the "locked pages" feature in SQL Server 2012*  *-- https://bit.ly/2F5UjOA*  *-- Memory Management Architecture Guide*  *-- https://bit.ly/2JKkadC*  *-- SQL Server Services information (Query 7) (SQL Server Services Info)*  **SELECT** servicename, process\_id, startup\_type\_desc, status\_desc,  last\_startup\_time, service\_account, is\_clustered, cluster\_nodename, [filename],  instant\_file\_initialization\_enabled  **FROM** sys.dm\_server\_services **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- Tells you the account being used for the SQL Server Service and the SQL Agent Service*  *-- Shows the process\_id, when they were last started, and their current status*  *-- Also shows whether you are running on a failover cluster instance, and what node you are running on*  *-- Also shows whether IFI is enabled*  *-- sys.dm\_server\_services (Transact-SQL)*  *-- https://bit.ly/2oKa1Un*  *-- Last backup information by database (Query 8) (Last Backup By Database)*  **SELECT** ISNULL(d.[name], bs.[database\_name]) **AS** [Database], d.recovery\_model\_desc **AS** [Recovery Model],  d.log\_reuse\_wait\_desc **AS** [Log Reuse Wait Desc],  CONVERT(**DECIMAL**(18,2), ds.cntr\_value/1024.0) **AS** [Total Data File Size on Disk (MB)],  CONVERT(**DECIMAL**(18,2), ls.cntr\_value/1024.0) **AS** [Total Log File Size on Disk (MB)],  CAST(CAST(lu.cntr\_value **AS** **FLOAT**) / CAST(ls.cntr\_value **AS** **FLOAT**) **AS** **DECIMAL**(18,2)) \* 100 **AS** [Log Used %],  MAX(**CASE** **WHEN** bs.[type] = 'D' **THEN** bs.backup\_finish\_date **ELSE** **NULL** **END**) **AS** [Last Full Backup],  MAX(**CASE** **WHEN** bs.[type] = 'D' **THEN** CONVERT (**BIGINT**, bs.compressed\_backup\_size / 1048576 ) **ELSE** **NULL** **END**) **AS** [Last Full Compressed Backup Size (MB)],  MAX(**CASE** **WHEN** bs.[type] = 'D' **THEN** CONVERT (**DECIMAL**(18,2), bs.backup\_size /bs.compressed\_backup\_size ) **ELSE** **NULL** **END**) **AS** [Backup Compression Ratio],  MAX(**CASE** **WHEN** bs.[type] = 'I' **THEN** bs.backup\_finish\_date **ELSE** **NULL** **END**) **AS** [Last Differential Backup],  MAX(**CASE** **WHEN** bs.[type] = 'L' **THEN** bs.backup\_finish\_date **ELSE** **NULL** **END**) **AS** [Last Log Backup],  DATABASEPROPERTYEX ((d.[name]), 'LastGoodCheckDbTime') **AS** [Last Good CheckDB]  **FROM** sys.databases **AS** d **WITH** (NOLOCK)  **INNER** **JOIN** sys.master\_files **as** mf **WITH** (NOLOCK)  **ON** d.database\_id = mf.database\_id  LEFT **OUTER** **JOIN** msdb.dbo.backupset **AS** bs **WITH** (NOLOCK)  **ON** bs.[database\_name] = d.[name]  **AND** bs.backup\_finish\_date > GETDATE()- 30  LEFT **OUTER** **JOIN** sys.dm\_os\_performance\_counters **AS** lu **WITH** (NOLOCK)  **ON** d.name = lu.instance\_name  LEFT **OUTER** **JOIN** sys.dm\_os\_performance\_counters **AS** ls **WITH** (NOLOCK)  **ON** d.name = ls.instance\_name  **INNER** **JOIN** sys.dm\_os\_performance\_counters **AS** ds **WITH** (NOLOCK)  **ON** d.name = ds.instance\_name  **WHERE** d.name <> N'tempdb'  **AND** lu.counter\_name **LIKE** N'Log File(s) Used Size (KB)%'  **AND** ls.counter\_name **LIKE** N'Log File(s) Size (KB)%'  **AND** ds.counter\_name **LIKE** N'Data File(s) Size (KB)%'  **AND** ls.cntr\_value > 0  **GROUP** **BY** ISNULL(d.[name], bs.[database\_name]), d.recovery\_model\_desc, d.log\_reuse\_wait\_desc, d.[name],  CONVERT(**DECIMAL**(18,2), ds.cntr\_value/1024.0),  CONVERT(**DECIMAL**(18,2), ls.cntr\_value/1024.0),  CAST(CAST(lu.cntr\_value **AS** **FLOAT**) / CAST(ls.cntr\_value **AS** **FLOAT**) **AS** **DECIMAL**(18,2)) \* 100  **ORDER** **BY** d.recovery\_model\_desc, d.[name] **OPTION** (RECOMPILE);  *------*  *-- This helps you spot runaway transaction logs and other issues with your backup schedule*  *-- Get SQL Server Agent jobs and Category information (Query 9) (SQL Server Agent Jobs)*  **SELECT** sj.name **AS** [Job Name], sj.[description] **AS** [Job Description],  sc.name **AS** [CategoryName], SUSER\_SNAME(sj.owner\_sid) **AS** [Job Owner],  sj.date\_created **AS** [Date Created], sj.[enabled] **AS** [Job Enabled],  sj.notify\_email\_operator\_id, sj.notify\_level\_email, h.run\_status,  RIGHT(STUFF(STUFF(REPLACE(STR(h.run\_duration, 7, 0), ' ', '0'), 4, 0, ':'), 7, 0, ':'),8) **AS** [Last Duration - HHMMSS],  CONVERT(**DATETIME**, RTRIM(h.run\_date) + ' ' + STUFF(STUFF(REPLACE(STR(RTRIM(h.run\_time),6,0),' ','0'),3,0,':'),6,0,':')) **AS** [Last Start Date]  **FROM** msdb.dbo.sysjobs **AS** sj **WITH** (NOLOCK)  LEFT **OUTER** **JOIN**  (**SELECT** job\_id, instance\_id = MAX(instance\_id)  **FROM** msdb.dbo.sysjobhistory **WITH** (NOLOCK)  **GROUP** **BY** job\_id) **AS** l  **ON** sj.job\_id = l.job\_id  LEFT **OUTER** **JOIN** msdb.dbo.syscategories **AS** sc **WITH** (NOLOCK)  **ON** sj.category\_id = sc.category\_id  LEFT **OUTER** **JOIN** msdb.dbo.sysjobhistory **AS** h **WITH** (NOLOCK)  **ON** h.job\_id = l.job\_id  **AND** h.instance\_id = l.instance\_id  **ORDER** **BY** CONVERT(**INT**, h.run\_duration) **DESC**, [Last Start Date] **DESC** **OPTION** (RECOMPILE);  *------*  *--run\_status*  *-- Value Status of the job execution*  *-- 0 = Failed*  *-- 1 = Succeeded*  *-- 2 = Retry*  *-- 3 = Canceled*  *-- 4 = In Progress*  *-- Gives you some basic information about your SQL Server Agent jobs, who owns them and how they are configured*  *-- Look for Agent jobs that are not owned by sa*  *-- Look for jobs that have a notify\_email\_operator\_id set to 0 (meaning no operator)*  *-- Look for jobs that have a notify\_level\_email set to 0 (meaning no e-mail is ever sent)*  *--*  *-- MSDN sysjobs documentation*  *-- https://bit.ly/2paDEOP*  *-- SQL Server Maintenance Solution (Ola Hallengren)*  *-- https://bit.ly/1pgchQu*  *-- You can use this script to add default schedules to the standard Ola Hallengren Maintenance Solution jobs*  *-- https://bit.ly/3ane0gN*  *-- Get SQL Server Agent Alert Information (Query 10) (SQL Server Agent Alerts)*  **SELECT** name, event\_source, message\_id, severity, [enabled], has\_notification,  delay\_between\_responses, occurrence\_count, last\_occurrence\_date, last\_occurrence\_time  **FROM** msdb.dbo.sysalerts **WITH** (NOLOCK)  **ORDER** **BY** name **OPTION** (RECOMPILE);  *------*  *-- Gives you some basic information about your SQL Server Agent Alerts*  *-- (which are different from SQL Server Agent jobs)*  *-- Read more about Agent Alerts here: https://bit.ly/2v5YR37*  *-- Host information (Query 11) (Host Info)*  **SELECT** host\_platform, host\_distribution, host\_release,  host\_service\_pack\_level, host\_sku, os\_language\_version,  host\_architecture  **FROM** sys.dm\_os\_host\_info **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- host\_release codes (only valid for Windows)*  *-- 10.0 is either Windows 10, Windows Server 2016 or Windows Server 2019*  *-- 6.3 is either Windows 8.1 or Windows Server 2012 R2*  *-- 6.2 is either Windows 8 or Windows Server 2012*  *-- host\_sku codes (only valid for Windows)*  *-- 4 is Enterprise Edition*  *-- 7 is Standard Server Edition*  *-- 8 is Datacenter Server Edition*  *-- 10 is Enterprise Server Edition*  *-- 48 is Professional Edition*  *-- 161 is Pro for Workstations*  *-- 1033 for os\_language\_version is US-English*  *-- SQL Server 2019 requires Windows Server 2016 or newer*  *-- Hardware and Software Requirements for Installing SQL Server*  *-- https://bit.ly/2y3ka5L*  *-- Using SQL Server in Windows 8 and later versions of Windows operating system*  *-- https://bit.ly/2F7Ax0P*  *-- SQL Server NUMA Node information (Query 12) (SQL Server NUMA Info)*  **SELECT** osn.node\_id, osn.node\_state\_desc, osn.memory\_node\_id, osn.processor\_group, osn.cpu\_count, osn.online\_scheduler\_count,  osn.idle\_scheduler\_count, osn.active\_worker\_count,  osmn.pages\_kb/1024 **AS** [Committed Memory (MB)],  osmn.locked\_page\_allocations\_kb/1024 **AS** [Locked Physical (MB)],  CONVERT(**DECIMAL**(18,2), osmn.foreign\_committed\_kb/1024.0) **AS** [Foreign Commited (MB)],  osmn.target\_kb/1024 **AS** [Target Memory Goal (MB)],  osn.avg\_load\_balance, osn.resource\_monitor\_state  **FROM** sys.dm\_os\_nodes **AS** osn **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_os\_memory\_nodes **AS** osmn **WITH** (NOLOCK)  **ON** osn.memory\_node\_id = osmn.memory\_node\_id  **WHERE** osn.node\_state\_desc <> N'ONLINE DAC' **OPTION** (RECOMPILE);  *------*  *-- Gives you some useful information about the composition and relative load on your NUMA nodes*  *-- You want to see an equal number of schedulers on each NUMA node*  *-- Watch out if SQL Server 2019 Standard Edition has been installed*  *-- on a physical or virtual machine with more than four sockets or more than 24 physical cores*  *-- sys.dm\_os\_nodes (Transact-SQL)*  *-- https://bit.ly/2pn5Mw8*  *-- How to Balance SQL Server Core Licenses Across NUMA Nodes*  *-- https://bit.ly/3i4TyVR*  *-- Good basic information about OS memory amounts and state (Query 13) (System Memory)*  **SELECT** total\_physical\_memory\_kb/1024 **AS** [Physical Memory (MB)],  available\_physical\_memory\_kb/1024 **AS** [Available Memory (MB)],  total\_page\_file\_kb/1024 **AS** [Page File Commit Limit (MB)],  total\_page\_file\_kb/1024 - total\_physical\_memory\_kb/1024 **AS** [Physical Page File Size (MB)],  available\_page\_file\_kb/1024 **AS** [Available Page File (MB)],  system\_cache\_kb/1024 **AS** [System Cache (MB)],  system\_memory\_state\_desc **AS** [System Memory State]  **FROM** sys.dm\_os\_sys\_memory **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- You want to see "Available physical memory is high" for System Memory State*  *-- This indicates that you are not under external memory pressure*  *-- Possible System Memory State values:*  *-- Available physical memory is high*  *-- Physical memory usage is steady*  *-- Available physical memory is low*  *-- Available physical memory is running low*  *-- Physical memory state is transitioning*  *-- sys.dm\_os\_sys\_memory (Transact-SQL)*  *-- https://bit.ly/2pcV0xq*  *-- You can skip the next two queries if you know you don't have a clustered instance*  *-- Get information about your cluster nodes and their status (Query 14) (Cluster Node Properties)*  *-- (if your database server is in a failover cluster)*  **SELECT** NodeName, status\_description, is\_current\_owner  **FROM** sys.dm\_os\_cluster\_nodes **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- Knowing which node owns the cluster resources is critical*  *-- Especially when you are installing Windows or SQL Server updates*  *-- You will see no results if your instance is not clustered*  *-- Recommended hotfixes and updates for Windows Server 2012 R2-based failover clusters*  *-- https://bit.ly/1z5BfCw*  *-- Get information about any AlwaysOn AG cluster this instance is a part of (Query 15) (AlwaysOn AG Cluster)*  **SELECT** cluster\_name, quorum\_type\_desc, quorum\_state\_desc  **FROM** sys.dm\_hadr\_cluster **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- You will see no results if your instance is not using AlwaysOn AGs*  *-- Good overview of AG health and status (Query 16) (AG Status)*  **SELECT** ag.name **AS** [AG Name], ar.replica\_server\_name, ar.availability\_mode\_desc, adc.[database\_name],  drs.is\_local, drs.is\_primary\_replica, drs.synchronization\_state\_desc, drs.is\_commit\_participant,  drs.synchronization\_health\_desc, drs.recovery\_lsn, drs.truncation\_lsn, drs.last\_sent\_lsn,  drs.last\_sent\_time, drs.last\_received\_lsn, drs.last\_received\_time, drs.last\_hardened\_lsn,  drs.last\_hardened\_time, drs.last\_redone\_lsn, drs.last\_redone\_time, drs.log\_send\_queue\_size,  drs.log\_send\_rate, drs.redo\_queue\_size, drs.redo\_rate, drs.filestream\_send\_rate,  drs.end\_of\_log\_lsn, drs.last\_commit\_lsn, drs.last\_commit\_time, drs.database\_state\_desc  **FROM** sys.dm\_hadr\_database\_replica\_states **AS** drs **WITH** (NOLOCK)  **INNER** **JOIN** sys.availability\_databases\_cluster **AS** adc **WITH** (NOLOCK)  **ON** drs.group\_id = adc.group\_id  **AND** drs.group\_database\_id = adc.group\_database\_id  **INNER** **JOIN** sys.availability\_groups **AS** ag **WITH** (NOLOCK)  **ON** ag.group\_id = drs.group\_id  **INNER** **JOIN** sys.availability\_replicas **AS** ar **WITH** (NOLOCK)  **ON** drs.group\_id = ar.group\_id  **AND** drs.replica\_id = ar.replica\_id  **ORDER** **BY** ag.name, ar.replica\_server\_name, adc.[database\_name] **OPTION** (RECOMPILE);  *-- You will see no results if your instance is not using AlwaysOn AGs*  *-- SQL Server 2016 It Just Runs Faster: Always On Availability Groups Turbocharged*  *-- https://bit.ly/2dn1H6r*  *-- Hardware information from SQL Server 2019 (Query 17) (Hardware Info)*  **SELECT** cpu\_count **AS** [Logical CPU Count], scheduler\_count,  (socket\_count \* cores\_per\_socket) **AS** [Physical Core Count],  socket\_count **AS** [Socket Count], cores\_per\_socket, numa\_node\_count,  physical\_memory\_kb/1024 **AS** [Physical Memory (MB)],  max\_workers\_count **AS** [Max Workers Count],  affinity\_type\_desc **AS** [Affinity Type],  sqlserver\_start\_time **AS** [SQL Server Start Time],  DATEDIFF(**hour**, sqlserver\_start\_time, GETDATE()) **AS** [SQL Server Up Time (hrs)],  virtual\_machine\_type\_desc **AS** [Virtual Machine Type],  softnuma\_configuration\_desc **AS** [Soft NUMA Configuration],  sql\_memory\_model\_desc,  container\_type\_desc *-- New in SQL Server 2019*  **FROM** sys.dm\_os\_sys\_info **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- Gives you some good basic hardware information about your database server*  *-- Note: virtual\_machine\_type\_desc of HYPERVISOR does not automatically mean you are running SQL Server inside of a VM*  *-- It merely indicates that you have a hypervisor running on your host*  *-- sys.dm\_os\_sys\_info (Transact-SQL)*  *-- https://bit.ly/2pczOYs*  *-- Soft NUMA configuration was a new column for SQL Server 2016*  *-- OFF = Soft-NUMA feature is OFF*  *-- ON = SQL Server automatically determines the NUMA node sizes for Soft-NUMA*  *-- MANUAL = Manually configured soft-NUMA*  *-- Configure SQL Server to Use Soft-NUMA (SQL Server)*  *-- https://bit.ly/2HTpKJt*  *-- sql\_memory\_model\_desc values (Added in SQL Server 2016 SP1)*  *-- CONVENTIONAL*  *-- LOCK\_PAGES*  *-- LARGE\_PAGES*    *-- Get System Manufacturer and model number from SQL Server Error log (Query 18) (System Manufacturer)*  **EXEC** sys.xp\_readerrorlog 0, 1, N'Manufacturer';  *------*  *-- This can help you determine the capabilities and capacities of your database server*  *-- Can also be used to confirm if you are running in a VM*  *-- This query might take a few seconds if you have not recycled your error log recently*  *-- This query will return no results if your error log has been recycled since the instance was started*  *-- Get BIOS date from Windows Registry (Query 19) (BIOS Date)*  **EXEC** sys.xp\_instance\_regread N'HKEY\_LOCAL\_MACHINE', N'HARDWARE\DESCRIPTION\System\BIOS', N'BiosReleaseDate';  *------*  *-- Helps you understand whether the main system BIOS is up to date, and the possible age of the hardware*  *-- Not as useful for virtualization*  *-- Does not work on Linux*  *-- Get processor description from Windows Registry (Query 20) (Processor Description)*  **EXEC** sys.xp\_instance\_regread N'HKEY\_LOCAL\_MACHINE', N'HARDWARE\DESCRIPTION\System\CentralProcessor\0', N'ProcessorNameString';  *------*  *-- Gives you the model number and rated clock speed of your processor(s)*  *-- Your processors may be running at less than the rated clock speed due*  *-- to the Windows Power Plan or hardware power management*  *-- Does not work on Linux*  *-- You can use CPU-Z to get your actual CPU core speed and a lot of other useful information*  *-- https://bit.ly/QhR6xF*  *-- You can learn more about processor selection for SQL Server by following this link*  *-- https://bit.ly/2F3aVlP*  *-- Get information on location, time and size of any memory dumps from SQL Server (Query 21) (Memory Dump Info)*  **SELECT** [filename], creation\_time, size\_in\_bytes/1048576.0 **AS** [Size (MB)]  **FROM** sys.dm\_server\_memory\_dumps **WITH** (NOLOCK)  **ORDER** **BY** creation\_time **DESC** **OPTION** (RECOMPILE);  *------*  *-- This will not return any rows if you have*  *-- not had any memory dumps (which is a good thing)*  *-- sys.dm\_server\_memory\_dumps (Transact-SQL)*  *-- https://bit.ly/2elwWll*  *-- Look at Suspect Pages table (Query 22) (Suspect Pages)*  **SELECT** DB\_NAME(sp.database\_id) **AS** [Database Name],  sp.[file\_id], sp.page\_id, sp.event\_type,  sp.error\_count, sp.last\_update\_date,  mf.name **AS** [Logical Name], mf.physical\_name **AS** [File Path]  **FROM** msdb.dbo.suspect\_pages **AS** sp **WITH** (NOLOCK)  **INNER** **JOIN** sys.master\_files **AS** mf **WITH** (NOLOCK)  **ON** mf.database\_id = sp.database\_id  **AND** mf.file\_id = sp.file\_id  **ORDER** **BY** sp.database\_id **OPTION** (RECOMPILE);  *------*  *-- event\_type value descriptions*  *-- 1 = 823 error caused by an operating system CRC error*  *-- or 824 error other than a bad checksum or a torn page (for example, a bad page ID)*  *-- 2 = Bad checksum*  *-- 3 = Torn page*  *-- 4 = Restored (The page was restored after it was marked bad)*  *-- 5 = Repaired (DBCC repaired the page)*  *-- 7 = Deallocated by DBCC*  *-- Ideally, this query returns no results. The table is limited to 1000 rows.*  *-- If you do get results here, you should do further investigation to determine the root cause*  *-- Manage the suspect\_pages Table*  *-- https://bit.ly/2Fvr1c9*  *-- Read most recent entries from all SQL Server Error Logs (Query 23) (Error Log Entries)*  **DROP** **TABLE** **IF** **EXISTS** #ErrorLogFiles;  **CREATE** **TABLE** #ErrorLogFiles  ([Archive #] **INT**,[Date] **NVARCHAR**(25),[Log File Size (Byte)]**INT**)  **INSERT** **INTO** #ErrorLogFiles  ([Archive #],[Date],[Log File Size (Byte)])  **EXEC** master.sys.xp\_enumerrorlogs;  **DROP** **TABLE** **IF** **EXISTS** #SQLErrorLog\_AllLogs;  **CREATE** **TABLE** #SQLErrorLog\_AllLogs  (LogDate **DATETIME** ,ProcessInfo **NVARCHAR**(12), LogText **NVARCHAR**(4000))  **DECLARE** @i **INT** = 0;  **DECLARE** @sql **NVARCHAR**(200) = N'';  **DECLARE** @logCount **INT** = (**SELECT** COUNT(\*) **FROM** #ErrorLogFiles);  **WHILE** (@i < @logCount)  **BEGIN**  **IF**(@i **in** (**SELECT** [Archive #] **FROM** #ErrorLogFiles))  **BEGIN**  **SET** @sql = N'INSERT INTO #SQLErrorLog\_AllLogs (LogDate, ProcessInfo, LogText)  EXEC master.sys.sp\_readerrorlog ' + CAST(@i **AS** **NVARCHAR**(2)) + N';'  **EXEC** master.sys.sp\_executesql @sql;  **END**  **SET** @i += 1;  **END**  **SELECT** **TOP**(1000)LogDate, ProcessInfo, LogText  **FROM** #SQLErrorLog\_AllLogs **WITH** (NOLOCK)  **ORDER** **BY** LogDate **DESC** **OPTION** (RECOMPILE);  **DROP** **TABLE** **IF** **EXISTS** #ErrorLogFiles;  **DROP** **TABLE** **IF** **EXISTS** #SQLErrorLog\_AllLogs;  **GO**  *------*  *-- Get number of data files in tempdb database (Query 24) (TempDB Data Files)*  **EXEC** sys.xp\_readerrorlog 0, 1, N'The tempdb database has';  *------*  *-- Get the number of data files in the tempdb database*  *-- 4-8 data files that are all the same size is a good starting point*  *-- This query will return no results if your error log has been recycled since the instance was last started*  *-- Find unequal tempdb data initial file sizes (Query 25) (Tempdb Data File Sizes)*  *-- This query might take a few seconds depending on the size of your error log*  **EXEC** sys.xp\_readerrorlog 0, 1, N'The tempdb database data files are not configured with the same initial size';  *------*  *-- You want this query to return no results*  *-- All of your tempdb data files should have the same initial size and autogrowth settings*  *-- This query will also return no results if your error log has been recycled since the instance was last started*  *-- KB3170020 - Informational messages added for tempdb configuration in the SQL Server error log in SQL Server 2012 and 2014*  *-- https://bit.ly/3IsR8jh*  *-- File names and paths for all user and system databases on instance (Query 26) (Database Filenames and Paths)*  **SELECT** DB\_NAME([database\_id]) **AS** [Database Name],  [file\_id], [name], physical\_name, [type\_desc], state\_desc,  is\_percent\_growth, growth,  CONVERT(**bigint**, growth/128.0) **AS** [Growth in MB],  CONVERT(**bigint**, **size**/128.0) **AS** [Total Size in MB], max\_size  **FROM** sys.master\_files **WITH** (NOLOCK)  **ORDER** **BY** DB\_NAME([database\_id]), [file\_id] **OPTION** (RECOMPILE);  *------*  *-- Things to look at:*  *-- Are data files and log files on different drives?*  *-- Is everything on the C: drive?*  *-- Is tempdb on dedicated drives?*  *-- Is there only one tempdb data file?*  *-- Are all of the tempdb data files the same size?*  *-- Are there multiple data files for user databases?*  *-- Is percent growth enabled for any files (which is bad)?*  *-- Drive information for all fixed drives visible to the operating system (Query 27) (Fixed Drives)*  **SELECT** fixed\_drive\_path, drive\_type\_desc,  CONVERT(**DECIMAL**(18,2), free\_space\_in\_bytes/1073741824.0) **AS** [Available Space (GB)]  **FROM** sys.dm\_os\_enumerate\_fixed\_drives **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- This shows all of your drives, not just LUNs with SQL Server database files*  *-- New in SQL Server 2017*  *-- sys.dm\_os\_enumerate\_fixed\_drives (Transact-SQL)*  *-- https://bit.ly/2EZoHLj*  *-- Volume info for all LUNS that have database files on the current instance (Query 28) (Volume Info)*  **SELECT** **DISTINCT** vs.volume\_mount\_point, vs.file\_system\_type, vs.logical\_volume\_name,  CONVERT(**DECIMAL**(18,2), vs.total\_bytes/1073741824.0) **AS** [Total Size (GB)],  CONVERT(**DECIMAL**(18,2), vs.available\_bytes/1073741824.0) **AS** [Available Size (GB)],  CONVERT(**DECIMAL**(18,2), vs.available\_bytes \* 1. / vs.total\_bytes \* 100.) **AS** [Space Free %],  vs.supports\_compression, vs.is\_compressed,  vs.supports\_sparse\_files, vs.supports\_alternate\_streams  **FROM** sys.master\_files **AS** f **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_os\_volume\_stats(f.database\_id, f.[file\_id]) **AS** vs  **ORDER** **BY** vs.volume\_mount\_point **OPTION** (RECOMPILE);  *------*  *-- Shows you the total and free space on the LUNs where you have database files*  *-- Being low on free space can negatively affect performance*  *-- sys.dm\_os\_volume\_stats (Transact-SQL)*  *-- https://bit.ly/2oBPNNr*  *-- Drive level latency information (Query 29) (Drive Level Latency)*  **SELECT** tab.[Drive], tab.volume\_mount\_point **AS** [Volume Mount Point],  **CASE**  **WHEN** num\_of\_reads = 0 **THEN** 0  **ELSE** (io\_stall\_read\_ms/num\_of\_reads)  **END** **AS** [Read Latency],  **CASE**  **WHEN** num\_of\_writes = 0 **THEN** 0  **ELSE** (io\_stall\_write\_ms/num\_of\_writes)  **END** **AS** [Write Latency],  **CASE**  **WHEN** (num\_of\_reads = 0 **AND** num\_of\_writes = 0) **THEN** 0  **ELSE** (io\_stall/(num\_of\_reads + num\_of\_writes))  **END** **AS** [Overall Latency],  **CASE**  **WHEN** num\_of\_reads = 0 **THEN** 0  **ELSE** (num\_of\_bytes\_read/num\_of\_reads)  **END** **AS** [Avg Bytes/Read],  **CASE**  **WHEN** num\_of\_writes = 0 **THEN** 0  **ELSE** (num\_of\_bytes\_written/num\_of\_writes)  **END** **AS** [Avg Bytes/Write],  **CASE**  **WHEN** (num\_of\_reads = 0 **AND** num\_of\_writes = 0) **THEN** 0  **ELSE** ((num\_of\_bytes\_read + num\_of\_bytes\_written)/(num\_of\_reads + num\_of\_writes))  **END** **AS** [Avg Bytes/Transfer]  **FROM** (**SELECT** LEFT(UPPER(mf.physical\_name), 2) **AS** Drive, SUM(num\_of\_reads) **AS** num\_of\_reads,  SUM(io\_stall\_read\_ms) **AS** io\_stall\_read\_ms, SUM(num\_of\_writes) **AS** num\_of\_writes,  SUM(io\_stall\_write\_ms) **AS** io\_stall\_write\_ms, SUM(num\_of\_bytes\_read) **AS** num\_of\_bytes\_read,  SUM(num\_of\_bytes\_written) **AS** num\_of\_bytes\_written, SUM(io\_stall) **AS** io\_stall, vs.volume\_mount\_point  **FROM** sys.dm\_io\_virtual\_file\_stats(**NULL**, **NULL**) **AS** vfs  **INNER** **JOIN** sys.master\_files **AS** mf **WITH** (NOLOCK)  **ON** vfs.database\_id = mf.database\_id **AND** vfs.file\_id = mf.file\_id  **CROSS** APPLY sys.dm\_os\_volume\_stats(mf.database\_id, mf.[file\_id]) **AS** vs  **GROUP** **BY** LEFT(UPPER(mf.physical\_name), 2), vs.volume\_mount\_point) **AS** tab  **ORDER** **BY** [Overall Latency] **OPTION** (RECOMPILE);  *------*  *-- Shows you the drive-level latency for reads and writes, in milliseconds*  *-- Latency above 30-40ms is usually a problem*  *-- These latency numbers include all file activity against all SQL Server*  *-- database files on each drive since SQL Server was last started*  *-- sys.dm\_io\_virtual\_file\_stats (Transact-SQL)*  *-- https://bit.ly/3bRWUc0*  *-- sys.dm\_os\_volume\_stats (Transact-SQL)*  *-- https://bit.ly/33thz2j*  *-- Calculates average latency per read, per write, and per total input/output for each database file (Query 30) (IO Latency by File)*  **SELECT** DB\_NAME(fs.database\_id) **AS** [Database Name], CAST(fs.io\_stall\_read\_ms/(1.0 + fs.num\_of\_reads) **AS** **NUMERIC**(10,1)) **AS** [avg\_read\_latency\_ms],  CAST(fs.io\_stall\_write\_ms/(1.0 + fs.num\_of\_writes) **AS** **NUMERIC**(10,1)) **AS** [avg\_write\_latency\_ms],  CAST((fs.io\_stall\_read\_ms + fs.io\_stall\_write\_ms)/(1.0 + fs.num\_of\_reads + fs.num\_of\_writes) **AS** **NUMERIC**(10,1)) **AS** [avg\_io\_latency\_ms],  CONVERT(**DECIMAL**(18,2), mf.**size**/128.0) **AS** [File Size (MB)], mf.physical\_name, mf.type\_desc, fs.io\_stall\_read\_ms, fs.num\_of\_reads,  fs.io\_stall\_write\_ms, fs.num\_of\_writes, fs.io\_stall\_read\_ms + fs.io\_stall\_write\_ms **AS** [io\_stalls], fs.num\_of\_reads + fs.num\_of\_writes **AS** [total\_io],  io\_stall\_queued\_read\_ms **AS** [Resource Governor Total Read IO Latency (ms)], io\_stall\_queued\_write\_ms **AS** [Resource Governor Total Write IO Latency (ms)]  **FROM** sys.dm\_io\_virtual\_file\_stats(**null**,**null**) **AS** fs  **INNER** **JOIN** sys.master\_files **AS** mf **WITH** (NOLOCK)  **ON** fs.database\_id = mf.database\_id  **AND** fs.[file\_id] = mf.[file\_id]  **ORDER** **BY** avg\_io\_latency\_ms **DESC** **OPTION** (RECOMPILE);  *------*  *-- Helps determine which database files on the entire instance have the most I/O bottlenecks*  *-- This can help you decide whether certain LUNs are overloaded and whether you might*  *-- want to move some files to a different location or perhaps improve your I/O performance*  *-- These latency numbers include all file activity against each SQL Server*  *-- database file since SQL Server was last started*  *-- sys.dm\_io\_virtual\_file\_stats (Transact-SQL)*  *-- https://bit.ly/3bRWUc0*  *-- Look for I/O requests taking longer than 15 seconds in the six most recent SQL Server Error Logs (Query 31) (IO Warnings)*  **CREATE** **TABLE** #IOWarningResults(LogDate **datetime**, ProcessInfo sysname, LogText **nvarchar**(1000));  **INSERT** **INTO** #IOWarningResults  **EXEC** xp\_readerrorlog 0, 1, N'taking longer than 15 seconds';  **INSERT** **INTO** #IOWarningResults  **EXEC** xp\_readerrorlog 1, 1, N'taking longer than 15 seconds';  **INSERT** **INTO** #IOWarningResults  **EXEC** xp\_readerrorlog 2, 1, N'taking longer than 15 seconds';  **INSERT** **INTO** #IOWarningResults  **EXEC** xp\_readerrorlog 3, 1, N'taking longer than 15 seconds';  **INSERT** **INTO** #IOWarningResults  **EXEC** xp\_readerrorlog 4, 1, N'taking longer than 15 seconds';  **INSERT** **INTO** #IOWarningResults  **EXEC** xp\_readerrorlog 5, 1, N'taking longer than 15 seconds';  **SELECT** LogDate, ProcessInfo, LogText  **FROM** #IOWarningResults  **ORDER** **BY** LogDate **DESC**;  **DROP** **TABLE** **IF** **EXISTS** #IOWarningResults;  *------*  *-- Finding 15 second I/O warnings in the SQL Server Error Log is useful evidence of*  *-- poor I/O performance (which might have many different causes)*  *-- Look to see if you see any patterns in the results (same files, same drives, same time of day, etc.)*  *-- Diagnostics in SQL Server help detect stalled and stuck I/O operations*  *-- https://bit.ly/2qtaw73*  *-- Resource Governor Resource Pool information (Query 32) (RG Resource Pools)*  **SELECT** pool\_id, [name], statistics\_start\_time,  min\_memory\_percent, max\_memory\_percent,  max\_memory\_kb/1024 **AS** [max\_memory\_mb],  used\_memory\_kb/1024 **AS** [used\_memory\_mb],  target\_memory\_kb/1024 **AS** [target\_memory\_mb],  min\_iops\_per\_volume, max\_iops\_per\_volume  **FROM** sys.dm\_resource\_governor\_resource\_pools **WITH** (NOLOCK)  **OPTION** (RECOMPILE);  *------*  *-- sys.dm\_resource\_governor\_resource\_pools (Transact-SQL)*  *-- https://bit.ly/2MVU0Vy*  *-- Recovery model, log reuse wait description, log file size, log usage size (Query 33) (Database Properties)*  *-- and compatibility level for all databases on instance*  **SELECT** db.[name] **AS** [Database Name], SUSER\_SNAME(db.owner\_sid) **AS** [Database Owner],  db.[compatibility\_level] **AS** [DB Compatibility Level],  db.recovery\_model\_desc **AS** [Recovery Model],  db.log\_reuse\_wait\_desc **AS** [Log Reuse Wait Description],  CONVERT(**DECIMAL**(18,2), ds.cntr\_value/1024.0) **AS** [Total Data File Size on Disk (MB)],  CONVERT(**DECIMAL**(18,2), ls.cntr\_value/1024.0) **AS** [Total Log File Size on Disk (MB)],  CONVERT(**DECIMAL**(18,2), lu.cntr\_value/1024.0) **AS** [Log File Used (MB)],  CAST(CAST(lu.cntr\_value **AS** **FLOAT**) / CAST(ls.cntr\_value **AS** **FLOAT**)**AS** **DECIMAL**(18,2)) \* 100 **AS** [Log Used %],  db.page\_verify\_option\_desc **AS** [Page Verify Option], db.user\_access\_desc, db.state\_desc, db.containment\_desc,  db.is\_mixed\_page\_allocation\_on,  db.is\_auto\_create\_stats\_on, db.is\_auto\_update\_stats\_on, db.is\_auto\_update\_stats\_async\_on, db.is\_parameterization\_forced,  db.snapshot\_isolation\_state\_desc, db.is\_read\_committed\_snapshot\_on, db.is\_auto\_close\_on, db.is\_auto\_shrink\_on,  db.target\_recovery\_time\_in\_seconds, db.is\_cdc\_enabled, db.is\_published, db.is\_distributor, db.is\_sync\_with\_backup,  db.group\_database\_id, db.replica\_id, db.is\_memory\_optimized\_enabled, db.is\_memory\_optimized\_elevate\_to\_snapshot\_on,  db.delayed\_durability\_desc, db.is\_query\_store\_on,  db.is\_temporal\_history\_retention\_enabled, db.is\_accelerated\_database\_recovery\_on,  db.is\_master\_key\_encrypted\_by\_server, db.is\_encrypted, de.encryption\_state, de.percent\_complete, de.key\_algorithm, de.key\_length  **FROM** sys.databases **AS** db **WITH** (NOLOCK)  LEFT **OUTER** **JOIN** sys.dm\_os\_performance\_counters **AS** lu **WITH** (NOLOCK)  **ON** db.name = lu.instance\_name  LEFT **OUTER** **JOIN** sys.dm\_os\_performance\_counters **AS** ls **WITH** (NOLOCK)  **ON** db.name = ls.instance\_name  LEFT **OUTER** **JOIN** sys.dm\_os\_performance\_counters **AS** ds **WITH** (NOLOCK)  **ON** db.name = ds.instance\_name  LEFT **OUTER** **JOIN** sys.dm\_database\_encryption\_keys **AS** de **WITH** (NOLOCK)  **ON** db.database\_id = de.database\_id  **WHERE** lu.counter\_name **LIKE** N'Log File(s) Used Size (KB)%'  **AND** ls.counter\_name **LIKE** N'Log File(s) Size (KB)%'  **AND** ds.counter\_name **LIKE** N'Data File(s) Size (KB)%'  **AND** ls.cntr\_value > 0  **ORDER** **BY** db.[name] **OPTION** (RECOMPILE);  *------*  *-- sys.databases (Transact-SQL)*  *-- https://bit.ly/2G5wqaX*  *-- sys.dm\_os\_performance\_counters (Transact-SQL)*  *-- https://bit.ly/3kEO2JR*  *-- sys.dm\_database\_encryption\_keys (Transact-SQL)*  *-- https://bit.ly/3mE7kkx*  *-- Things to look at:*  *-- How many databases are on the instance?*  *-- What recovery models are they using?*  *-- What is the log reuse wait description?*  *-- How full are the transaction logs?*  *-- What compatibility level are the databases on?*  *-- What is the Page Verify Option? (should be CHECKSUM)*  *-- Is Auto Update Statistics Asynchronously enabled?*  *-- What is target\_recovery\_time\_in\_seconds? (should be 60 for user databases)*  *-- Is Delayed Durability enabled?*  *-- Make sure auto\_shrink and auto\_close are not enabled!*  *-- is\_mixed\_page\_allocation\_on is a new property for SQL Server 2016. Equivalent to TF 1118 for a user database*  *-- SQL Server 2016: Changes in default behavior for autogrow and allocations for tempdb and user databases*  *-- https://bit.ly/2evRZSR*  *-- A non-zero value for target\_recovery\_time\_in\_seconds means that indirect checkpoint is enabled*  *-- If the setting has a zero value it indicates that automatic checkpoint is enabled*  *-- Changes in SQL Server 2016 Checkpoint Behavior*  *-- https://bit.ly/2pdggk3*  *-- Missing Indexes for all databases by Index Advantage (Query 34) (Missing Indexes All Databases)*  **SELECT** CONVERT(**decimal**(18,2), migs.user\_seeks \* migs.avg\_total\_user\_cost \* (migs.avg\_user\_impact \* 0.01)) **AS** [index\_advantage],  CONVERT(**nvarchar**(25), migs.last\_user\_seek, 20) **AS** [last\_user\_seek],  mid.[statement] **AS** [Database.Schema.Table],  COUNT(1) **OVER**(**PARTITION** **BY** mid.[statement]) **AS** [missing\_indexes\_for\_table],  COUNT(1) **OVER**(**PARTITION** **BY** mid.[statement], mid.equality\_columns) **AS** [similar\_missing\_indexes\_for\_table],  mid.equality\_columns, mid.inequality\_columns, mid.included\_columns, migs.user\_seeks,  CONVERT(**decimal**(18,2), migs.avg\_total\_user\_cost) **AS** [avg\_total\_user\_,cost], migs.avg\_user\_impact,  REPLACE(REPLACE(LEFT(st.[text], 255), **CHAR**(10),''), **CHAR**(13),'') **AS** [Short Query Text]  **FROM** sys.dm\_db\_missing\_index\_groups **AS** mig **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_db\_missing\_index\_group\_stats\_query **AS** migs **WITH**(NOLOCK)  **ON** mig.index\_group\_handle = migs.group\_handle  **CROSS** APPLY sys.dm\_exec\_sql\_text(migs.last\_sql\_handle) **AS** st  **INNER** **JOIN** sys.dm\_db\_missing\_index\_details **AS** mid **WITH** (NOLOCK)  **ON** mig.index\_handle = mid.index\_handle  **ORDER** **BY** index\_advantage **DESC** **OPTION** (RECOMPILE);  *------*  *-- Getting missing index information for all of the databases on the instance is very useful*  *-- Look at last user seek time, number of user seeks to help determine source and importance*  *-- Also look at avg\_user\_impact and avg\_total\_user\_cost to help determine importance*  *-- SQL Server is overly eager to add included columns, so beware*  *-- Do not just blindly add indexes that show up from this query!!!*  *-- Hkan Winther has given me some great suggestions for this query*  *-- SQL Server Index Design Guide*  *-- https://bit.ly/2qtZr4N*  *-- Get VLF Counts for all databases on the instance (Query 35) (VLF Counts)*  **SELECT** db.[name] **AS** [Database Name], li.[VLF Count]  **FROM** sys.databases **AS** db **WITH** (NOLOCK)  **CROSS** APPLY (**SELECT** file\_id, COUNT(\*) **AS** [VLF Count]  **FROM** sys.dm\_db\_log\_info (db.database\_id)  **GROUP** **BY** file\_id) **AS** li  **ORDER** **BY** li.[VLF Count] **DESC** **OPTION** (RECOMPILE);  *------*  *-- High VLF counts can affect write performance to the log file*  *-- and they can make full database restores and crash recovery take much longer*  *-- Try to keep your VLF counts under 200 in most cases (depending on log file size)*  *-- sys.dm\_db\_log\_info (Transact-SQL)*  *-- https://bit.ly/3jpmqsd*  *-- sys.databases (Transact-SQL)*  *-- https://bit.ly/2G5wqaX*  *-- SQL Server Transaction Log Architecture and Management Guide*  *-- https://bit.ly/2JjmQRZ*  *-- VLF Growth Formula (SQL Server 2014 and newer)*  *-- If the log growth increment is less than 1/8th the current size of the log*  *-- Then: 1 new VLF*  *-- Otherwise:*  *-- Up to 64MB: 4 new VLFs*  *-- 64MB to 1GB: 8 new VLFs*  *-- More than 1GB: 16 new VLFs*  *-- Get CPU utilization by database (Query 36) (CPU Usage by Database)*  **WITH** DB\_CPU\_Stats  **AS**  (**SELECT** pa.DatabaseID, DB\_Name(pa.DatabaseID) **AS** [Database Name], SUM(qs.total\_worker\_time/1000) **AS** [CPU\_Time\_Ms]  **FROM** sys.dm\_exec\_query\_stats **AS** qs **WITH** (NOLOCK)  **CROSS** APPLY (**SELECT** CONVERT(**int**, **value**) **AS** [DatabaseID]  **FROM** sys.dm\_exec\_plan\_attributes(qs.plan\_handle)  **WHERE** attribute = N'dbid') **AS** pa  **GROUP** **BY** DatabaseID)  **SELECT** ROW\_NUMBER() **OVER**(**ORDER** **BY** [CPU\_Time\_Ms] **DESC**) **AS** [CPU Rank],  [Database Name], [CPU\_Time\_Ms] **AS** [CPU Time (ms)],  CAST([CPU\_Time\_Ms] \* 1.0 / SUM([CPU\_Time\_Ms]) **OVER**() \* 100.0 **AS** **DECIMAL**(5, 2)) **AS** [CPU Percent]  **FROM** DB\_CPU\_Stats  **WHERE** DatabaseID <> 32767 *-- ResourceDB*  **ORDER** **BY** [CPU Rank] **OPTION** (RECOMPILE);  *------*  *-- Helps determine which database is using the most CPU resources on the instance*  *-- Note: This only reflects CPU usage from the currently cached query plans*  *-- sys.dm\_exec\_query\_stats (Transact-SQL)*  *-- https://bit.ly/32tHCGH*  *-- sys.dm\_exec\_plan\_attributes (Transact-SQL)*  *-- https://bit.ly/35iP2hV*  *-- Get I/O utilization by database (Query 37) (IO Usage By Database)*  **WITH** Aggregate\_IO\_Statistics  **AS** (**SELECT** DB\_NAME(database\_id) **AS** [Database Name],  CAST(SUM(num\_of\_bytes\_read + num\_of\_bytes\_written) / 1048576 **AS** **DECIMAL**(12, 2)) **AS** [ioTotalMB],  CAST(SUM(num\_of\_bytes\_read ) / 1048576 **AS** **DECIMAL**(12, 2)) **AS** [ioReadMB],  CAST(SUM(num\_of\_bytes\_written) / 1048576 **AS** **DECIMAL**(12, 2)) **AS** [ioWriteMB]  **FROM** sys.dm\_io\_virtual\_file\_stats(**NULL**, **NULL**) **AS** [DM\_IO\_STATS]  **GROUP** **BY** database\_id)  **SELECT** ROW\_NUMBER() **OVER** (**ORDER** **BY** ioTotalMB **DESC**) **AS** [I/O Rank],  [Database Name], ioTotalMB **AS** [Total I/O (MB)],  CAST(ioTotalMB / SUM(ioTotalMB) **OVER** () \* 100.0 **AS** **DECIMAL**(5, 2)) **AS** [Total I/O %],  ioReadMB **AS** [Read I/O (MB)],  CAST(ioReadMB / SUM(ioReadMB) **OVER** () \* 100.0 **AS** **DECIMAL**(5, 2)) **AS** [Read I/O %],  ioWriteMB **AS** [Write I/O (MB)],  CAST(ioWriteMB / SUM(ioWriteMB) **OVER** () \* 100.0 **AS** **DECIMAL**(5, 2)) **AS** [Write I/O %]  **FROM** Aggregate\_IO\_Statistics  **ORDER** **BY** [I/O Rank] **OPTION** (RECOMPILE);  *------*  *-- Helps determine which database is using the most I/O resources on the instance*  *-- These numbers are cumulative since the last service restart*  *-- They include all I/O activity, not just the nominal I/O workload*  *-- sys.dm\_io\_virtual\_file\_stats (Transact-SQL)*  *-- https://bit.ly/3bRWUc0*  *-- Get total buffer usage by database for current instance (Query 38) (Total Buffer Usage by Database)*  *-- This may take some time to run on a busy instance with lots of RAM*  **WITH** AggregateBufferPoolUsage  **AS**  (**SELECT** DB\_NAME(database\_id) **AS** [Database Name],  CAST(COUNT\_BIG(\*) \* 8/1024.0 **AS** **DECIMAL** (15,2)) **AS** [CachedSize],  COUNT(page\_id) **AS** [Page Count],  AVG(read\_microsec) **AS** [Avg Read Time (microseconds)]  **FROM** sys.dm\_os\_buffer\_descriptors **WITH** (NOLOCK)  **GROUP** **BY** DB\_NAME(database\_id))  **SELECT** ROW\_NUMBER() **OVER**(**ORDER** **BY** CachedSize **DESC**) **AS** [Buffer Pool Rank], [Database Name],  CAST(CachedSize / SUM(CachedSize) **OVER**() \* 100.0 **AS** **DECIMAL**(5,2)) **AS** [Buffer Pool Percent],  [Page Count], CachedSize **AS** [Cached Size (MB)], [Avg Read Time (microseconds)]  **FROM** AggregateBufferPoolUsage  **ORDER** **BY** [Buffer Pool Rank] **OPTION** (RECOMPILE);  *------*  *-- Tells you how much memory (in the buffer pool)*  *-- is being used by each database on the instance*  *-- sys.dm\_os\_buffer\_descriptors (Transact-SQL)*  *-- https://bit.ly/36s7aFo*  *-- Get tempdb version store space usage by database (Query 39) (Version Store Space Usage)*  **SELECT** DB\_NAME(database\_id) **AS** [Database Name],  reserved\_page\_count **AS** [Version Store Reserved Page Count],  reserved\_space\_kb/1024 **AS** [Version Store Reserved Space (MB)]  **FROM** sys.dm\_tran\_version\_store\_space\_usage **WITH** (NOLOCK)  **ORDER** **BY** reserved\_space\_kb/1024 **DESC** **OPTION** (RECOMPILE);  *------*  *-- sys.dm\_tran\_version\_store\_space\_usage (Transact-SQL)*  *-- https://bit.ly/2vh3Bmk*  *-- Clear Wait Stats with this command*  *-- DBCC SQLPERF('sys.dm\_os\_wait\_stats', CLEAR);*  *-- Isolate top waits for server instance since last restart or wait statistics clear (Query 40) (Top Waits)*  **WITH** [Waits]  **AS** (**SELECT** wait\_type, wait\_time\_ms/ 1000.0 **AS** [WaitS],  (wait\_time\_ms - signal\_wait\_time\_ms) / 1000.0 **AS** [ResourceS],  signal\_wait\_time\_ms / 1000.0 **AS** [SignalS],  waiting\_tasks\_count **AS** [WaitCount],  100.0 \* wait\_time\_ms / SUM (wait\_time\_ms) **OVER**() **AS** [Percentage],  ROW\_NUMBER() **OVER**(**ORDER** **BY** wait\_time\_ms **DESC**) **AS** [RowNum]  **FROM** sys.dm\_os\_wait\_stats **WITH** (NOLOCK)  **WHERE** [wait\_type] **NOT** **IN** (  N'BROKER\_EVENTHANDLER', N'BROKER\_RECEIVE\_WAITFOR', N'BROKER\_TASK\_STOP',  N'BROKER\_TO\_FLUSH', N'BROKER\_TRANSMITTER', N'CHECKPOINT\_QUEUE',  N'CHKPT', N'CLR\_AUTO\_EVENT', N'CLR\_MANUAL\_EVENT', N'CLR\_SEMAPHORE', N'CXCONSUMER',  N'DBMIRROR\_DBM\_EVENT', N'DBMIRROR\_EVENTS\_QUEUE', N'DBMIRROR\_WORKER\_QUEUE',  N'DBMIRRORING\_CMD', N'DIRTY\_PAGE\_POLL', N'DISPATCHER\_QUEUE\_SEMAPHORE',  N'EXECSYNC', N'FSAGENT', N'FT\_IFTS\_SCHEDULER\_IDLE\_WAIT', N'FT\_IFTSHC\_MUTEX',  N'HADR\_CLUSAPI\_CALL', N'HADR\_FILESTREAM\_IOMGR\_IOCOMPLETION', N'HADR\_LOGCAPTURE\_WAIT',  N'HADR\_NOTIFICATION\_DEQUEUE', N'HADR\_TIMER\_TASK', N'HADR\_WORK\_QUEUE',  N'KSOURCE\_WAKEUP', N'LAZYWRITER\_SLEEP', N'LOGMGR\_QUEUE',  N'MEMORY\_ALLOCATION\_EXT', N'ONDEMAND\_TASK\_QUEUE',  N'PARALLEL\_REDO\_DRAIN\_WORKER', N'PARALLEL\_REDO\_LOG\_CACHE', N'PARALLEL\_REDO\_TRAN\_LIST',  N'PARALLEL\_REDO\_WORKER\_SYNC', N'PARALLEL\_REDO\_WORKER\_WAIT\_WORK',  N'PREEMPTIVE\_COM\_GETDATA', N'PREEMPTIVE\_COM\_QUERYINTERFACE',  N'PREEMPTIVE\_HADR\_LEASE\_MECHANISM', N'PREEMPTIVE\_SP\_SERVER\_DIAGNOSTICS',  N'PREEMPTIVE\_OS\_LIBRARYOPS', N'PREEMPTIVE\_OS\_COMOPS', N'PREEMPTIVE\_OS\_CRYPTOPS',  N'PREEMPTIVE\_OS\_PIPEOPS', N'PREEMPTIVE\_OS\_AUTHENTICATIONOPS',  N'PREEMPTIVE\_OS\_GENERICOPS', N'PREEMPTIVE\_OS\_VERIFYTRUST',  N'PREEMPTIVE\_OS\_FILEOPS', N'PREEMPTIVE\_OS\_DEVICEOPS', N'PREEMPTIVE\_OS\_QUERYREGISTRY',  N'PREEMPTIVE\_OS\_WRITEFILE', N'PREEMPTIVE\_OS\_WRITEFILEGATHER',  N'PREEMPTIVE\_XE\_CALLBACKEXECUTE', N'PREEMPTIVE\_XE\_DISPATCHER',  N'PREEMPTIVE\_XE\_GETTARGETSTATE', N'PREEMPTIVE\_XE\_SESSIONCOMMIT',  N'PREEMPTIVE\_XE\_TARGETINIT', N'PREEMPTIVE\_XE\_TARGETFINALIZE',  N'PWAIT\_ALL\_COMPONENTS\_INITIALIZED', N'PWAIT\_DIRECTLOGCONSUMER\_GETNEXT',  N'PWAIT\_EXTENSIBILITY\_CLEANUP\_TASK',  N'QDS\_PERSIST\_TASK\_MAIN\_LOOP\_SLEEP', N'QDS\_ASYNC\_QUEUE',  N'QDS\_CLEANUP\_STALE\_QUERIES\_TASK\_MAIN\_LOOP\_SLEEP', N'REQUEST\_FOR\_DEADLOCK\_SEARCH',  N'RESOURCE\_QUEUE', N'SERVER\_IDLE\_CHECK', N'SLEEP\_BPOOL\_FLUSH', N'SLEEP\_DBSTARTUP',  N'SLEEP\_DCOMSTARTUP', N'SLEEP\_MASTERDBREADY', N'SLEEP\_MASTERMDREADY',  N'SLEEP\_MASTERUPGRADED', N'SLEEP\_MSDBSTARTUP', N'SLEEP\_SYSTEMTASK', N'SLEEP\_TASK',  N'SLEEP\_TEMPDBSTARTUP', N'SNI\_HTTP\_ACCEPT', N'SOS\_WORK\_DISPATCHER',  N'SP\_SERVER\_DIAGNOSTICS\_SLEEP', N'SOS\_WORKER\_MIGRATION', N'VDI\_CLIENT\_OTHER',  N'SQLTRACE\_BUFFER\_FLUSH', N'SQLTRACE\_INCREMENTAL\_FLUSH\_SLEEP', N'SQLTRACE\_WAIT\_ENTRIES',  N'STARTUP\_DEPENDENCY\_MANAGER',  N'WAIT\_FOR\_RESULTS', N'WAITFOR', N'WAITFOR\_TASKSHUTDOWN', N'WAIT\_XTP\_HOST\_WAIT',  N'WAIT\_XTP\_OFFLINE\_CKPT\_NEW\_LOG', N'WAIT\_XTP\_CKPT\_CLOSE', N'WAIT\_XTP\_RECOVERY',  N'XE\_BUFFERMGR\_ALLPROCESSED\_EVENT', N'XE\_DISPATCHER\_JOIN',  N'XE\_DISPATCHER\_WAIT', N'XE\_LIVE\_TARGET\_TVF', N'XE\_TIMER\_EVENT')  **AND** waiting\_tasks\_count > 0)  **SELECT**  MAX (W1.wait\_type) **AS** [WaitType],  CAST (MAX (W1.Percentage) **AS** **DECIMAL** (5,2)) **AS** [Wait Percentage],  CAST ((MAX (W1.WaitS) / MAX (W1.WaitCount)) **AS** **DECIMAL** (16,4)) **AS** [AvgWait\_Sec],  CAST ((MAX (W1.ResourceS) / MAX (W1.WaitCount)) **AS** **DECIMAL** (16,4)) **AS** [AvgRes\_Sec],  CAST ((MAX (W1.SignalS) / MAX (W1.WaitCount)) **AS** **DECIMAL** (16,4)) **AS** [AvgSig\_Sec],  CAST (MAX (W1.WaitS) **AS** **DECIMAL** (16,2)) **AS** [Wait\_Sec],  CAST (MAX (W1.ResourceS) **AS** **DECIMAL** (16,2)) **AS** [Resource\_Sec],  CAST (MAX (W1.SignalS) **AS** **DECIMAL** (16,2)) **AS** [Signal\_Sec],  MAX (W1.WaitCount) **AS** [Wait Count],  CAST (N'https://www.sqlskills.com/help/waits/' + W1.wait\_type **AS** **XML**) **AS** [Help/Info URL]  **FROM** Waits **AS** W1  **INNER** **JOIN** Waits **AS** W2  **ON** W2.RowNum <= W1.RowNum  **GROUP** **BY** W1.RowNum, W1.wait\_type  **HAVING** SUM (W2.Percentage) - MAX (W1.Percentage) < 99 *-- percentage threshold*  **OPTION** (RECOMPILE);  *------*  *-- Cumulative wait stats are not as useful on an idle instance that is not under load or performance pressure*  *-- SQL Server Wait Types Library*  *-- https://bit.ly/2ePzYO2*  *-- The SQL Server Wait Type Repository*  *-- https://bit.ly/1afzfjC*  *-- Wait statistics, or please tell me where it hurts*  *-- https://bit.ly/2wsQHQE*  *-- SQL Server 2005 Performance Tuning using the Waits and Queues*  *-- https://bit.ly/1o2NFoF*  *-- sys.dm\_os\_wait\_stats (Transact-SQL)*  *-- https://bit.ly/2Hjq9Yl*  *-- Get a count of SQL connections by IP address (Query 41) (Connection Counts by IP Address)*  **SELECT** ec.client\_net\_address, es.[program\_name], es.[host\_name], es.login\_name,  COUNT(ec.session\_id) **AS** [connection count]  **FROM** sys.dm\_exec\_sessions **AS** es **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_exec\_connections **AS** ec **WITH** (NOLOCK)  **ON** es.session\_id = ec.session\_id  **GROUP** **BY** ec.client\_net\_address, es.[program\_name], es.[host\_name], es.login\_name  **ORDER** **BY** ec.client\_net\_address, es.[program\_name] **OPTION** (RECOMPILE);  *------*  *-- This helps you figure where your database load is coming from*  *-- and verifies connectivity from other machines*  *-- Solving Connectivity errors to SQL Server*  *-- https://bit.ly/2EgzoD0*  *-- Get Average Task Counts (run multiple times) (Query 42) (Avg Task Counts)*  **SELECT** AVG(current\_tasks\_count) **AS** [Avg Task Count],  AVG(work\_queue\_count) **AS** [Avg Work Queue Count],  AVG(runnable\_tasks\_count) **AS** [Avg Runnable Task Count],  AVG(pending\_disk\_io\_count) **AS** [Avg Pending DiskIO Count],  GETDATE() **AS** [System Time]  **FROM** sys.dm\_os\_schedulers **WITH** (NOLOCK)  **WHERE** scheduler\_id < 255 **OPTION** (RECOMPILE);  *------*  *-- Sustained values above 10 suggest further investigation in that area*  *-- High Avg Task Counts are often caused by blocking/deadlocking or other resource contention*  *-- Sustained values above 1 suggest further investigation in that area*  *-- High Avg Runnable Task Counts are a good sign of CPU pressure*  *-- High Avg Pending DiskIO Counts are a sign of disk pressure*  *-- How to Do Some Very Basic SQL Server Monitoring*  *-- https://bit.ly/30IRla0*  *-- Detect blocking (run multiple times) (Query 43) (Detect Blocking)*  **SELECT** t1.resource\_type **AS** [lock type], DB\_NAME(resource\_database\_id) **AS** [database],  t1.resource\_associated\_entity\_id **AS** [blk object],t1.request\_mode **AS** [lock req], *-- lock requested*  t1.request\_session\_id **AS** [waiter sid], t2.wait\_duration\_ms **AS** [wait time], *-- spid of waiter*  (**SELECT** [text] **FROM** sys.dm\_exec\_requests **AS** r **WITH** (NOLOCK) *-- get sql for waiter*  **CROSS** APPLY sys.dm\_exec\_sql\_text(r.[sql\_handle])  **WHERE** r.session\_id = t1.request\_session\_id) **AS** [waiter\_batch],  (**SELECT** SUBSTRING(qt.[text],r.statement\_start\_offset/2,  (**CASE** **WHEN** r.statement\_end\_offset = -1  **THEN** LEN(CONVERT(**NVARCHAR**(max), qt.[text])) \* 2  **ELSE** r.statement\_end\_offset **END** - r.statement\_start\_offset)/2)  **FROM** sys.dm\_exec\_requests **AS** r **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_sql\_text(r.[sql\_handle]) **AS** qt  **WHERE** r.session\_id = t1.request\_session\_id) **AS** [waiter\_stmt], *-- statement blocked*  t2.blocking\_session\_id **AS** [blocker sid], *-- spid of blocker*  (**SELECT** [text] **FROM** sys.sysprocesses **AS** p *-- get sql for blocker*  **CROSS** APPLY sys.dm\_exec\_sql\_text(p.[sql\_handle])  **WHERE** p.spid = t2.blocking\_session\_id) **AS** [blocker\_batch]  **FROM** sys.dm\_tran\_locks **AS** t1 **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_os\_waiting\_tasks **AS** t2 **WITH** (NOLOCK)  **ON** t1.lock\_owner\_address = t2.resource\_address **OPTION** (RECOMPILE);  *------*  *-- Helps troubleshoot blocking and deadlocking issues*  *-- The results will change from second to second on a busy system*  *-- You should run this query multiple times when you see signs of blocking*  *-- Show page level contention (Query 44) (Page Contention)*  **SELECT** er.session\_id, er.wait\_type, er.wait\_resource,  OBJECT\_NAME(pinfo.[object\_id], pinfo.database\_id) **AS** [object\_name],  er.blocking\_session\_id, er.command,  SUBSTRING(st.**text**, (er.statement\_start\_offset/2)+1,  ((**CASE** er.statement\_end\_offset  **WHEN** -1 **THEN** DATALENGTH(st.**text**)  **ELSE** er.statement\_end\_offset  **END** - er.statement\_start\_offset)/2) + 1) **AS** statement\_text,  DB\_NAME(pinfo.database\_id) **AS** [Database Name],  pinfo.[file\_id], pinfo.page\_id, pinfo.[object\_id], pinfo.index\_id, pinfo.page\_type\_desc  **FROM** sys.dm\_exec\_requests **AS** er **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_sql\_text(er.sql\_handle) **AS** st  **CROSS** APPLY sys.fn\_PageResCracker(er.page\_resource) **AS** r  **CROSS** APPLY sys.dm\_db\_page\_info(r.[db\_id], r.[file\_id], r.page\_id, N'DETAILED') **AS** pinfo  **WHERE** er.wait\_type **LIKE** N'%page%' **OPTION** (RECOMPILE);  *------*  *-- sys.fn\_PageResCracker (Transact-SQL)*  *-- https://bit.ly/3sgwp9B*  *-- Get CPU Utilization History for last 256 minutes (in one minute intervals) (Query 45) (CPU Utilization History)*  **DECLARE** @ts\_now **bigint** = (**SELECT** ms\_ticks **FROM** sys.dm\_os\_sys\_info **WITH** (NOLOCK));  **SELECT** **TOP**(256) SQLProcessUtilization **AS** [SQL Server Process CPU Utilization],  SystemIdle **AS** [System Idle Process],  100 - SystemIdle - SQLProcessUtilization **AS** [Other Process CPU Utilization],  DATEADD(ms, -1 \* (@ts\_now - [timestamp]), GETDATE()) **AS** [Event Time]  **FROM** (**SELECT** record.**value**('(./Record/@id)[1]', 'int') **AS** record\_id,  record.**value**('(./Record/SchedulerMonitorEvent/SystemHealth/SystemIdle)[1]', 'int')  **AS** [SystemIdle],  record.**value**('(./Record/SchedulerMonitorEvent/SystemHealth/ProcessUtilization)[1]', 'int')  **AS** [SQLProcessUtilization], [timestamp]  **FROM** (**SELECT** [timestamp], CONVERT(**xml**, record) **AS** [record]  **FROM** sys.dm\_os\_ring\_buffers **WITH** (NOLOCK)  **WHERE** ring\_buffer\_type = N'RING\_BUFFER\_SCHEDULER\_MONITOR'  **AND** record **LIKE** N'%<SystemHealth>%') **AS** x) **AS** y  **ORDER** **BY** record\_id **DESC** **OPTION** (RECOMPILE);  *------*  *-- Look at the trend over the entire period*  *-- Also look at high sustained 'Other Process' CPU Utilization values*  *-- Note: This query sometimes gives inaccurate results (negative values)*  *-- on high core count (> 64 cores) systems*  *-- Get top total worker time queries for entire instance (Query 46) (Top Worker Time Queries)*  **SELECT** **TOP**(50) DB\_NAME(t.[dbid]) **AS** [Database Name],  REPLACE(REPLACE(LEFT(t.[text], 255), **CHAR**(10),''), **CHAR**(13),'') **AS** [Short Query Text],  qs.total\_worker\_time **AS** [Total Worker Time], qs.min\_worker\_time **AS** [Min Worker Time],  qs.total\_worker\_time/qs.execution\_count **AS** [Avg Worker Time],  qs.max\_worker\_time **AS** [Max Worker Time],  qs.min\_elapsed\_time **AS** [Min Elapsed Time],  qs.total\_elapsed\_time/qs.execution\_count **AS** [Avg Elapsed Time],  qs.max\_elapsed\_time **AS** [Max Elapsed Time],  qs.min\_logical\_reads **AS** [Min Logical Reads],  qs.total\_logical\_reads/qs.execution\_count **AS** [Avg Logical Reads],  qs.max\_logical\_reads **AS** [Max Logical Reads],  qs.execution\_count **AS** [Execution Count],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  qs.creation\_time **AS** [Creation Time]  *--,t.[text] AS [Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel*  **FROM** sys.dm\_exec\_query\_stats **AS** qs **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_sql\_text(plan\_handle) **AS** t  **CROSS** APPLY sys.dm\_exec\_query\_plan(plan\_handle) **AS** qp  **ORDER** **BY** qs.total\_worker\_time **DESC** **OPTION** (RECOMPILE);  *------*  *-- Helps you find the most expensive queries from a CPU perspective across the entire instance*  *-- Can also help track down parameter sniffing issues*  *-- Page Life Expectancy (PLE) value for each NUMA node in current instance (Query 47) (PLE by NUMA Node)*  **SELECT** @@SERVERNAME **AS** [Server Name], RTRIM([object\_name]) **AS** [Object Name],  instance\_name, cntr\_value **AS** [Page Life Expectancy], GETDATE() **AS** [System Time]  **FROM** sys.dm\_os\_performance\_counters **WITH** (NOLOCK)  **WHERE** [object\_name] **LIKE** N'%Buffer Node%' *-- Handles named instances*  **AND** counter\_name = N'Page life expectancy' **OPTION** (RECOMPILE);  *------*  *-- PLE is a good measurement of internal memory pressure*  *-- Higher PLE is better. Watch the trend over time, not the absolute value*  *-- This will only return one row for non-NUMA systems*  *-- Page Life Expectancy isnt what you think*  *-- https://bit.ly/2EgynLa*  *-- Memory Grants Pending value for current instance (Query 48) (Memory Grants Pending)*  **SELECT** @@SERVERNAME **AS** [Server Name], RTRIM([object\_name]) **AS** [Object Name], cntr\_value **AS** [Memory Grants Pending]  **FROM** sys.dm\_os\_performance\_counters **WITH** (NOLOCK)  **WHERE** [object\_name] **LIKE** N'%Memory Manager%' *-- Handles named instances*  **AND** counter\_name = N'Memory Grants Pending' **OPTION** (RECOMPILE);  *------*  *-- Run multiple times, and run periodically if you suspect you are under memory pressure*  *-- Memory Grants Pending above zero for a sustained period is a very strong indicator of internal memory pressure*  *-- Memory Clerk Usage for instance (Query 49) (Memory Clerk Usage)*  *-- Look for high value for CACHESTORE\_SQLCP (Ad-hoc query plans)*  **SELECT** **TOP**(10) mc.[type] **AS** [Memory Clerk Type],  CAST((SUM(mc.pages\_kb)/1024.0) **AS** **DECIMAL** (15,2)) **AS** [Memory Usage (MB)]  **FROM** sys.dm\_os\_memory\_clerks **AS** mc **WITH** (NOLOCK)  **GROUP** **BY** mc.[type]  **ORDER** **BY** SUM(mc.pages\_kb) **DESC** **OPTION** (RECOMPILE);  *------*  *-- MEMORYCLERK\_SQLBUFFERPOOL was new for SQL Server 2012. It should be your highest consumer of memory*  *-- CACHESTORE\_SQLCP - SQL Plans*  *-- These are cached SQL statements or batches that aren't in stored procedures, functions and triggers*  *-- Watch out for high values for CACHESTORE\_SQLCP*  *-- Enabling 'optimize for ad hoc workloads' at the instance level can help reduce this*  *-- Running DBCC FREESYSTEMCACHE ('SQL Plans'); periodically may be required to better control this*  *-- CACHESTORE\_OBJCP - Object Plans*  *-- These are compiled plans for stored procedures, functions and triggers*  *-- If you see very high usage by MEMORYCLERK\_SQLLOGPOOL*  *-- SQL Server 2019 CU9 added a new command, DBCC FREESYSTEMCACHE ('LogPool');*  *-- sys.dm\_os\_memory\_clerks (Transact-SQL)*  *-- https://bit.ly/2H31xDR*  *-- Find single-use, ad-hoc and prepared queries that are bloating the plan cache (Query 50) (Ad hoc Queries)*  **SELECT** **TOP**(50) DB\_NAME(t.[dbid]) **AS** [Database Name],  REPLACE(REPLACE(LEFT(t.[text], 255), **CHAR**(10),''), **CHAR**(13),'') **AS** [Short Query Text],  cp.objtype **AS** [Object Type], cp.cacheobjtype **AS** [Cache Object Type],  cp.size\_in\_bytes/1024 **AS** [Plan Size in KB],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index]  *--,t.[text] AS [Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel*  **FROM** sys.dm\_exec\_cached\_plans **AS** cp **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_sql\_text(plan\_handle) **AS** t  **CROSS** APPLY sys.dm\_exec\_query\_plan(plan\_handle) **AS** qp  **WHERE** cp.cacheobjtype = N'Compiled Plan'  **AND** cp.objtype **IN** (N'Adhoc', N'Prepared')  **AND** cp.usecounts = 1  **ORDER** **BY** cp.size\_in\_bytes **DESC**, DB\_NAME(t.[dbid]) **OPTION** (RECOMPILE);  *------*  *-- Gives you the text, type and size of single-use ad-hoc and prepared queries that waste space in the plan cache*  *-- Enabling 'optimize for ad hoc workloads' for the instance can help (SQL Server 2008 and above only)*  *-- Running DBCC FREESYSTEMCACHE ('SQL Plans') periodically may be required to better control this*  *-- Enabling forced parameterization for the database can help, but test first!*  *-- Plan cache, adhoc workloads and clearing the single-use plan cache bloat*  *-- https://bit.ly/2EfYOkl*  *-- Get top total logical reads queries for entire instance (Query 51) (Top Logical Reads Queries)*  **SELECT** **TOP**(50) DB\_NAME(t.[dbid]) **AS** [Database Name],  REPLACE(REPLACE(LEFT(t.[text], 255), **CHAR**(10),''), **CHAR**(13),'') **AS** [Short Query Text],  qs.total\_logical\_reads **AS** [Total Logical Reads],  qs.min\_logical\_reads **AS** [Min Logical Reads],  qs.total\_logical\_reads/qs.execution\_count **AS** [Avg Logical Reads],  qs.max\_logical\_reads **AS** [Max Logical Reads],  qs.min\_worker\_time **AS** [Min Worker Time],  qs.total\_worker\_time/qs.execution\_count **AS** [Avg Worker Time],  qs.max\_worker\_time **AS** [Max Worker Time],  qs.min\_elapsed\_time **AS** [Min Elapsed Time],  qs.total\_elapsed\_time/qs.execution\_count **AS** [Avg Elapsed Time],  qs.max\_elapsed\_time **AS** [Max Elapsed Time],  qs.execution\_count **AS** [Execution Count],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  qs.creation\_time **AS** [Creation Time]  *--,t.[text] AS [Complete Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel*  **FROM** sys.dm\_exec\_query\_stats **AS** qs **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_sql\_text(plan\_handle) **AS** t  **CROSS** APPLY sys.dm\_exec\_query\_plan(plan\_handle) **AS** qp  **ORDER** **BY** qs.total\_logical\_reads **DESC** **OPTION** (RECOMPILE);  *------*  *-- Helps you find the most expensive queries from a memory perspective across the entire instance*  *-- Can also help track down parameter sniffing issues*  *-- Get top average elapsed time queries for entire instance (Query 52) (Top Avg Elapsed Time Queries)*  **SELECT** **TOP**(50) DB\_NAME(t.[dbid]) **AS** [Database Name],  REPLACE(REPLACE(LEFT(t.[text], 255), **CHAR**(10),''), **CHAR**(13),'') **AS** [Short Query Text],  qs.total\_elapsed\_time/qs.execution\_count **AS** [Avg Elapsed Time],  qs.min\_elapsed\_time, qs.max\_elapsed\_time, qs.last\_elapsed\_time,  qs.execution\_count **AS** [Execution Count],  qs.total\_logical\_reads/qs.execution\_count **AS** [Avg Logical Reads],  qs.total\_physical\_reads/qs.execution\_count **AS** [Avg Physical Reads],  qs.total\_worker\_time/qs.execution\_count **AS** [Avg Worker Time],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  qs.creation\_time **AS** [Creation Time]  *--,t.[text] AS [Complete Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel*  **FROM** sys.dm\_exec\_query\_stats **AS** qs **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_sql\_text(plan\_handle) **AS** t  **CROSS** APPLY sys.dm\_exec\_query\_plan(plan\_handle) **AS** qp  **ORDER** **BY** qs.total\_elapsed\_time/qs.execution\_count **DESC** **OPTION** (RECOMPILE);  *------*  *-- Helps you find the highest average elapsed time queries across the entire instance*  *-- Can also help track down parameter sniffing issues*  *-- Look at UDF execution statistics (Query 53) (UDF Stats by DB)*  **SELECT** **TOP** (25) DB\_NAME(database\_id) **AS** [Database Name],  OBJECT\_NAME(object\_id, database\_id) **AS** [Function Name],  total\_worker\_time, execution\_count, total\_elapsed\_time,  total\_elapsed\_time/execution\_count **AS** [avg\_elapsed\_time],  last\_elapsed\_time, last\_execution\_time, cached\_time, [type\_desc]  **FROM** sys.dm\_exec\_function\_stats **WITH** (NOLOCK)  **ORDER** **BY** total\_worker\_time **DESC** **OPTION** (RECOMPILE);  *------*  *-- sys.dm\_exec\_function\_stats (Transact-SQL)*  *-- https://bit.ly/2q1Q6BM*  *-- Showplan Enhancements for UDFs*  *-- https://bit.ly/2LVqiQ1*  *-- Look for long duration buffer pool scans (Query 54) (Long Buffer Pool Scans)*  **EXEC** sys.xp\_readerrorlog 0, 1, N'Buffer pool scan took';  *------*  *-- Finds buffer pool scans that took more than 10 seconds in the current SQL Server Error log*  *-- Only in SQL Server 2019 CU9 and later*  *-- Operations that trigger buffer pool scan may run slowly on large-memory computers - SQL Server | Microsoft Docs*  *-- https://bit.ly/3QrFC81*  *-- Database specific queries \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *-- \*\*\*\* Please switch to a user database that you are interested in! \*\*\*\*\**  *--USE YourDatabaseName; -- make sure to change to an actual database on your instance, not the master system database*  *--GO*  *-- Individual File Sizes and space available for current database (Query 55) (File Sizes and Space)*  **SELECT** f.[name] **AS** [File Name] , f.physical\_name **AS** [Physical Name],  CAST((f.**size**/128.0) **AS** **DECIMAL**(15,2)) **AS** [Total Size in MB],  CAST((f.**size**/128.0) **AS** **DECIMAL**(15,2)) -  CAST(f.**size**/128.0 - CAST(FILEPROPERTY(f.name, 'SpaceUsed') **AS** **int**)/128.0 **AS** **DECIMAL**(15,2))  **AS** [Used Space in MB],  CAST(f.**size**/128.0 - CAST(FILEPROPERTY(f.name, 'SpaceUsed') **AS** **int**)/128.0 **AS** **DECIMAL**(15,2))  **AS** [Available Space In MB],  f.[file\_id], fg.name **AS** [Filegroup Name],  f.is\_percent\_growth, f.growth, fg.is\_default, fg.is\_read\_only, fg.is\_autogrow\_all\_files  **FROM** sys.database\_files **AS** f **WITH** (NOLOCK)  LEFT **OUTER** **JOIN** sys.filegroups **AS** fg **WITH** (NOLOCK)  **ON** f.data\_space\_id = fg.data\_space\_id  **ORDER** **BY** f.[type], f.[file\_id] **OPTION** (RECOMPILE);  *------*  *-- Look at how large and how full the files are and where they are located*  *-- Make sure the transaction log is not full!!*  *-- is\_autogrow\_all\_files was new for SQL Server 2016. Equivalent to TF 1117 for user databases*  *-- SQL Server 2016: Changes in default behavior for autogrow and allocations for tempdb and user databases*  *-- https://bit.ly/2evRZSR*  *-- Log space usage for current database (Query 56) (Log Space Usage)*  **SELECT** DB\_NAME(lsu.database\_id) **AS** [Database Name], db.recovery\_model\_desc **AS** [Recovery Model],  CAST(lsu.total\_log\_size\_in\_bytes/1048576.0 **AS** **DECIMAL**(10, 2)) **AS** [Total Log Space (MB)],  CAST(lsu.used\_log\_space\_in\_bytes/1048576.0 **AS** **DECIMAL**(10, 2)) **AS** [Used Log Space (MB)],  CAST(lsu.used\_log\_space\_in\_percent **AS** **DECIMAL**(10, 2)) **AS** [Used Log Space %],  CAST(lsu.log\_space\_in\_bytes\_since\_last\_backup/1048576.0 **AS** **DECIMAL**(10, 2)) **AS** [Used Log Space Since Last Backup (MB)],  db.log\_reuse\_wait\_desc  **FROM** sys.dm\_db\_log\_space\_usage **AS** lsu **WITH** (NOLOCK)  **INNER** **JOIN** sys.databases **AS** db **WITH** (NOLOCK)  **ON** lsu.database\_id = db.database\_id  **OPTION** (RECOMPILE);  *------*  *-- Look at log file size and usage, along with the log reuse wait description for the current database*  *-- sys.dm\_db\_log\_space\_usage (Transact-SQL)*  *-- https://bit.ly/2H4MQw9*  *-- Status of last VLF for current database (Query 57) (Last VLF Status)*  **SELECT** **TOP**(1) DB\_NAME(li.database\_id) **AS** [Database Name], li.[file\_id],  li.vlf\_size\_mb, li.vlf\_sequence\_number, li.vlf\_active, li.vlf\_status  **FROM** sys.dm\_db\_log\_info(DB\_ID()) **AS** li  **ORDER** **BY** vlf\_sequence\_number **DESC** **OPTION** (RECOMPILE);  *------*  *-- Determine whether you will be able to shrink the transaction log file*  *-- vlf\_status Values*  *-- 0 is inactive*  *-- 1 is initialized but unused*  *-- 2 is active*  *-- sys.dm\_db\_log\_info (Transact-SQL)*  *-- https://bit.ly/2EQUU1v*  *-- Get database scoped configuration values for current database (Query 58) (Database-scoped Configurations)*  **SELECT** configuration\_id, name, [value] **AS** [value\_for\_primary], value\_for\_secondary, is\_value\_default  **FROM** sys.database\_scoped\_configurations **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- This lets you see the value of these new properties for the current database*  *-- Clear plan cache for current database*  *-- ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE\_CACHE;*  *-- ALTER DATABASE SCOPED CONFIGURATION (Transact-SQL)*  *-- https://bit.ly/2sOH7nb*  *-- I/O Statistics by file for the current database (Query 59) (IO Stats By File)*  **SELECT** DB\_NAME(DB\_ID()) **AS** [Database Name], df.name **AS** [Logical Name], vfs.[file\_id], df.type\_desc,  df.physical\_name **AS** [Physical Name], CAST(vfs.size\_on\_disk\_bytes/1048576.0 **AS** **DECIMAL**(15, 2)) **AS** [Size on Disk (MB)],  vfs.num\_of\_reads, vfs.num\_of\_writes, vfs.io\_stall\_read\_ms, vfs.io\_stall\_write\_ms,  CAST(100. \* vfs.io\_stall\_read\_ms/(vfs.io\_stall\_read\_ms + vfs.io\_stall\_write\_ms) **AS** **DECIMAL**(10,1)) **AS** [IO Stall Reads Pct],  CAST(100. \* vfs.io\_stall\_write\_ms/(vfs.io\_stall\_write\_ms + vfs.io\_stall\_read\_ms) **AS** **DECIMAL**(10,1)) **AS** [IO Stall Writes Pct],  (vfs.num\_of\_reads + vfs.num\_of\_writes) **AS** [Writes + Reads],  CAST(vfs.num\_of\_bytes\_read/1048576.0 **AS** **DECIMAL**(15, 2)) **AS** [MB Read],  CAST(vfs.num\_of\_bytes\_written/1048576.0 **AS** **DECIMAL**(15, 2)) **AS** [MB Written],  CAST(100. \* vfs.num\_of\_reads/(vfs.num\_of\_reads + vfs.num\_of\_writes) **AS** **DECIMAL**(15,1)) **AS** [# Reads Pct],  CAST(100. \* vfs.num\_of\_writes/(vfs.num\_of\_reads + vfs.num\_of\_writes) **AS** **DECIMAL**(15,1)) **AS** [# Write Pct],  CAST(100. \* vfs.num\_of\_bytes\_read/(vfs.num\_of\_bytes\_read + vfs.num\_of\_bytes\_written) **AS** **DECIMAL**(15,1)) **AS** [Read Bytes Pct],  CAST(100. \* vfs.num\_of\_bytes\_written/(vfs.num\_of\_bytes\_read + vfs.num\_of\_bytes\_written) **AS** **DECIMAL**(15,1)) **AS** [Written Bytes Pct]  **FROM** sys.dm\_io\_virtual\_file\_stats(DB\_ID(), **NULL**) **AS** vfs  **INNER** **JOIN** sys.database\_files **AS** df **WITH** (NOLOCK)  **ON** vfs.[file\_id]= df.[file\_id] **OPTION** (RECOMPILE);  *------*  *-- This helps you characterize your workload better from an I/O perspective for this database*  *-- It helps you determine whether you have an OLTP or DW/DSS type of workload*  *-- Get most frequently executed queries for this database (Query 60) (Query Execution Counts)*  **SELECT** **TOP**(50) LEFT(t.[text], 50) **AS** [Short Query Text], qs.execution\_count **AS** [Execution Count],  ISNULL(qs.execution\_count/DATEDIFF(**Minute**, qs.creation\_time, GETDATE()), 0) **AS** [Calls/Minute],  qs.total\_logical\_reads **AS** [Total Logical Reads],  qs.total\_logical\_reads/qs.execution\_count **AS** [Avg Logical Reads],  qs.total\_worker\_time **AS** [Total Worker Time],  qs.total\_worker\_time/qs.execution\_count **AS** [Avg Worker Time],  qs.total\_elapsed\_time **AS** [Total Elapsed Time],  qs.total\_elapsed\_time/qs.execution\_count **AS** [Avg Elapsed Time],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  qs.last\_execution\_time **AS** [Last Execution Time], qs.creation\_time **AS** [Creation Time]  *--,t.[text] AS [Complete Query Text], qp.query\_plan AS [Query Plan] -- uncomment out these columns if not copying results to Excel*  **FROM** sys.dm\_exec\_query\_stats **AS** qs **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_sql\_text(plan\_handle) **AS** t  **CROSS** APPLY sys.dm\_exec\_query\_plan(plan\_handle) **AS** qp  **WHERE** t.dbid = DB\_ID()  **AND** DATEDIFF(**Minute**, qs.creation\_time, GETDATE()) > 0  **ORDER** **BY** qs.execution\_count **DESC** **OPTION** (RECOMPILE);  *------*  *-- Tells you which cached queries are called the most often*  *-- This helps you characterize and baseline your workload*  *-- It also helps you find possible caching opportunities*  *-- CREATE PROCEDURE (Transact-SQL)*  *-- https://bit.ly/3gxcuxG*  *-- Queries 61 through 67 are the "Bad Man List" for stored procedures*  *-- Top Cached SPs By Execution Count (Query 61) (SP Execution Counts)*  **SELECT** **TOP**(100) p.name **AS** [SP Name], qs.execution\_count **AS** [Execution Count],  ISNULL(qs.execution\_count/DATEDIFF(**Minute**, qs.cached\_time, GETDATE()), 0) **AS** [Calls/Minute],  qs.total\_elapsed\_time/qs.execution\_count **AS** [Avg Elapsed Time],  qs.total\_worker\_time/qs.execution\_count **AS** [Avg Worker Time],  qs.total\_logical\_reads/qs.execution\_count **AS** [Avg Logical Reads],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  CONVERT(**nvarchar**(25), qs.last\_execution\_time, 20) **AS** [Last Execution Time],  CONVERT(**nvarchar**(25), qs.cached\_time, 20) **AS** [Plan Cached Time]  *-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan*  **FROM** sys.procedures **AS** p **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_exec\_procedure\_stats **AS** qs **WITH** (NOLOCK)  **ON** p.[object\_id] = qs.[object\_id]  **CROSS** APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) **AS** qp  **WHERE** qs.database\_id = DB\_ID()  **AND** DATEDIFF(**Minute**, qs.cached\_time, GETDATE()) > 0  **ORDER** **BY** qs.execution\_count **DESC** **OPTION** (RECOMPILE);  *------*  *-- Tells you which cached stored procedures are called the most often*  *-- This helps you characterize and baseline your workload*  *-- It also helps you find possible caching opportunities*  *-- Top Cached SPs By Avg Elapsed Time (Query 62) (SP Avg Elapsed Time)*  **SELECT** **TOP**(25) p.name **AS** [SP Name], qs.min\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count **AS** [avg\_elapsed\_time],  qs.max\_elapsed\_time, qs.last\_elapsed\_time, qs.total\_elapsed\_time, qs.execution\_count,  ISNULL(qs.execution\_count/DATEDIFF(**Minute**, qs.cached\_time, GETDATE()), 0) **AS** [Calls/Minute],  qs.total\_worker\_time/qs.execution\_count **AS** [AvgWorkerTime],  qs.total\_worker\_time **AS** [TotalWorkerTime],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  CONVERT(**nvarchar**(25), qs.last\_execution\_time, 20) **AS** [Last Execution Time],  CONVERT(**nvarchar**(25), qs.cached\_time, 20) **AS** [Plan Cached Time]  *-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan*  **FROM** sys.procedures **AS** p **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_exec\_procedure\_stats **AS** qs **WITH** (NOLOCK)  **ON** p.[object\_id] = qs.[object\_id]  **CROSS** APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) **AS** qp  **WHERE** qs.database\_id = DB\_ID()  **AND** DATEDIFF(**Minute**, qs.cached\_time, GETDATE()) > 0  **ORDER** **BY** avg\_elapsed\_time **DESC** **OPTION** (RECOMPILE);  *------*  *-- This helps you find high average elapsed time cached stored procedures that*  *-- may be easy to optimize with standard query tuning techniques*  *-- Top Cached SPs By Total Worker time. Worker time relates to CPU cost (Query 63) (SP Worker Time)*  **SELECT** **TOP**(25) p.name **AS** [SP Name], qs.total\_worker\_time **AS** [TotalWorkerTime],  qs.total\_worker\_time/qs.execution\_count **AS** [AvgWorkerTime], qs.execution\_count,  ISNULL(qs.execution\_count/DATEDIFF(**Minute**, qs.cached\_time, GETDATE()), 0) **AS** [Calls/Minute],  qs.total\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count **AS** [avg\_elapsed\_time],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  CONVERT(**nvarchar**(25), qs.last\_execution\_time, 20) **AS** [Last Execution Time],  CONVERT(**nvarchar**(25), qs.cached\_time, 20) **AS** [Plan Cached Time]  *--,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan*  **FROM** sys.procedures **AS** p **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_exec\_procedure\_stats **AS** qs **WITH** (NOLOCK)  **ON** p.[object\_id] = qs.[object\_id]  **CROSS** APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) **AS** qp  **WHERE** qs.database\_id = DB\_ID()  **AND** DATEDIFF(**Minute**, qs.cached\_time, GETDATE()) > 0  **ORDER** **BY** qs.total\_worker\_time **DESC** **OPTION** (RECOMPILE);  *------*  *-- This helps you find the most expensive cached stored procedures from a CPU perspective*  *-- You should look at this if you see signs of CPU pressure*  *-- Top Cached SPs By Total Logical Reads. Logical reads relate to memory pressure (Query 64) (SP Logical Reads)*  **SELECT** **TOP**(25) p.name **AS** [SP Name], qs.total\_logical\_reads **AS** [TotalLogicalReads],  qs.total\_logical\_reads/qs.execution\_count **AS** [AvgLogicalReads],qs.execution\_count,  ISNULL(qs.execution\_count/DATEDIFF(**Minute**, qs.cached\_time, GETDATE()), 0) **AS** [Calls/Minute],  qs.total\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count **AS** [avg\_elapsed\_time],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  CONVERT(**nvarchar**(25), qs.last\_execution\_time, 20) **AS** [Last Execution Time],  CONVERT(**nvarchar**(25), qs.cached\_time, 20) **AS** [Plan Cached Time]  *-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan*  **FROM** sys.procedures **AS** p **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_exec\_procedure\_stats **AS** qs **WITH** (NOLOCK)  **ON** p.[object\_id] = qs.[object\_id]  **CROSS** APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) **AS** qp  **WHERE** qs.database\_id = DB\_ID()  **AND** DATEDIFF(**Minute**, qs.cached\_time, GETDATE()) > 0  **ORDER** **BY** qs.total\_logical\_reads **DESC** **OPTION** (RECOMPILE);  *------*  *-- This helps you find the most expensive cached stored procedures from a memory perspective*  *-- You should look at this if you see signs of memory pressure*  *-- Top Cached SPs By Total Physical Reads. Physical reads relate to disk read I/O pressure (Query 65) (SP Physical Reads)*  **SELECT** **TOP**(25) p.name **AS** [SP Name],qs.total\_physical\_reads **AS** [TotalPhysicalReads],  qs.total\_physical\_reads/qs.execution\_count **AS** [AvgPhysicalReads], qs.execution\_count,  qs.total\_logical\_reads,qs.total\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count **AS** [avg\_elapsed\_time],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  CONVERT(**nvarchar**(25), qs.last\_execution\_time, 20) **AS** [Last Execution Time],  CONVERT(**nvarchar**(25), qs.cached\_time, 20) **AS** [Plan Cached Time]  *-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan*  **FROM** sys.procedures **AS** p **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_exec\_procedure\_stats **AS** qs **WITH** (NOLOCK)  **ON** p.[object\_id] = qs.[object\_id]  **CROSS** APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) **AS** qp  **WHERE** qs.database\_id = DB\_ID()  **AND** qs.total\_physical\_reads > 0  **ORDER** **BY** qs.total\_physical\_reads **DESC**, qs.total\_logical\_reads **DESC** **OPTION** (RECOMPILE);  *------*  *-- This helps you find the most expensive cached stored procedures from a read I/O perspective*  *-- You should look at this if you see signs of I/O pressure or of memory pressure*    *-- Top Cached SPs By Total Logical Writes (Query 66) (SP Logical Writes)*  *-- Logical writes relate to both memory and disk I/O pressure*  **SELECT** **TOP**(25) p.name **AS** [SP Name], qs.total\_logical\_writes **AS** [TotalLogicalWrites],  qs.total\_logical\_writes/qs.execution\_count **AS** [AvgLogicalWrites], qs.execution\_count,  ISNULL(qs.execution\_count/DATEDIFF(**Minute**, qs.cached\_time, GETDATE()), 0) **AS** [Calls/Minute],  qs.total\_elapsed\_time, qs.total\_elapsed\_time/qs.execution\_count **AS** [avg\_elapsed\_time],  **CASE** **WHEN** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%' **THEN** 1 **ELSE** 0 **END** **AS** [Has Missing Index],  CONVERT(**nvarchar**(25), qs.last\_execution\_time, 20) **AS** [Last Execution Time],  CONVERT(**nvarchar**(25), qs.cached\_time, 20) **AS** [Plan Cached Time]  *-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan*  **FROM** sys.procedures **AS** p **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_exec\_procedure\_stats **AS** qs **WITH** (NOLOCK)  **ON** p.[object\_id] = qs.[object\_id]  **CROSS** APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) **AS** qp  **WHERE** qs.database\_id = DB\_ID()  **AND** qs.total\_logical\_writes > 0  **AND** DATEDIFF(**Minute**, qs.cached\_time, GETDATE()) > 0  **ORDER** **BY** qs.total\_logical\_writes **DESC** **OPTION** (RECOMPILE);  *------*  *-- This helps you find the most expensive cached stored procedures from a write I/O perspective*  *-- You should look at this if you see signs of I/O pressure or of memory pressure*  *-- Cached SPs Missing Indexes by Execution Count (Query 67) (SP Missing Index)*  **SELECT** **TOP**(25) p.name **AS** [SP Name], qs.execution\_count **AS** [Execution Count],  ISNULL(qs.execution\_count/DATEDIFF(**Minute**, qs.cached\_time, GETDATE()), 0) **AS** [Calls/Minute],  qs.total\_elapsed\_time/qs.execution\_count **AS** [Avg Elapsed Time],  qs.total\_worker\_time/qs.execution\_count **AS** [Avg Worker Time],  qs.total\_logical\_reads/qs.execution\_count **AS** [Avg Logical Reads],  CONVERT(**nvarchar**(25), qs.last\_execution\_time, 20) **AS** [Last Execution Time],  CONVERT(**nvarchar**(25), qs.cached\_time, 20) **AS** [Plan Cached Time]  *-- ,qp.query\_plan AS [Query Plan] -- Uncomment if you want the Query Plan*  **FROM** sys.procedures **AS** p **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_exec\_procedure\_stats **AS** qs **WITH** (NOLOCK)  **ON** p.[object\_id] = qs.[object\_id]  **CROSS** APPLY sys.dm\_exec\_query\_plan(qs.plan\_handle) **AS** qp  **WHERE** qs.database\_id = DB\_ID()  **AND** DATEDIFF(**Minute**, qs.cached\_time, GETDATE()) > 0  **AND** CONVERT(**nvarchar**(max), qp.query\_plan) **COLLATE** Latin1\_General\_BIN2 **LIKE** N'%<MissingIndexes>%'  **ORDER** **BY** qs.execution\_count **DESC** **OPTION** (RECOMPILE);  *------*  *-- This helps you find the most frequently executed cached stored procedures that have missing index warnings*  *-- This can often help you find index tuning candidates*  *-- Lists the top statements by average input/output usage for the current database (Query 68) (Top IO Statements)*  **SELECT** **TOP**(50) OBJECT\_NAME(qt.objectid, dbid) **AS** [SP Name],  (qs.total\_logical\_reads + qs.total\_logical\_writes) /qs.execution\_count **AS** [Avg IO], qs.execution\_count **AS** [Execution Count],  SUBSTRING(qt.[text],qs.statement\_start\_offset/2,  (**CASE**  **WHEN** qs.statement\_end\_offset = -1  **THEN** LEN(CONVERT(**nvarchar**(max), qt.[text])) \* 2  **ELSE** qs.statement\_end\_offset  **END** - qs.statement\_start\_offset)/2) **AS** [Query Text]  **FROM** sys.dm\_exec\_query\_stats **AS** qs **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_sql\_text(qs.sql\_handle) **AS** qt  **WHERE** qt.[dbid] = DB\_ID()  **ORDER** **BY** [Avg IO] **DESC** **OPTION** (RECOMPILE);  *------*  *-- Helps you find the most expensive statements for I/O by SP*  *-- Possible Bad NC Indexes (writes > reads) (Query 69) (Bad NC Indexes)*  **SELECT** SCHEMA\_NAME(o.[schema\_id]) **AS** [Schema Name],  OBJECT\_NAME(s.[object\_id]) **AS** [Table Name],  i.name **AS** [Index Name], i.index\_id,  i.is\_disabled, i.is\_hypothetical, i.has\_filter, i.fill\_factor,  s.user\_updates **AS** [Total Writes], s.user\_seeks + s.user\_scans + s.user\_lookups **AS** [Total Reads],  s.user\_updates - (s.user\_seeks + s.user\_scans + s.user\_lookups) **AS** [Difference]  **FROM** sys.dm\_db\_index\_usage\_stats **AS** s **WITH** (NOLOCK)  **INNER** **JOIN** sys.indexes **AS** i **WITH** (NOLOCK)  **ON** s.[object\_id] = i.[object\_id]  **AND** i.index\_id = s.index\_id  **INNER** **JOIN** sys.objects **AS** o **WITH** (NOLOCK)  **ON** i.[object\_id] = o.[object\_id]  **WHERE** OBJECTPROPERTY(s.[object\_id],'IsUserTable') = 1  **AND** s.database\_id = DB\_ID()  **AND** s.user\_updates > (s.user\_seeks + s.user\_scans + s.user\_lookups)  **AND** i.index\_id > 1 **AND** i.[type\_desc] = N'NONCLUSTERED'  **AND** i.is\_primary\_key = 0 **AND** i.is\_unique\_constraint = 0 **AND** i.is\_unique = 0  **ORDER** **BY** [Difference] **DESC**, [Total Writes] **DESC**, [Total Reads] **ASC** **OPTION** (RECOMPILE);  *------*  *-- Look for indexes with high numbers of writes and zero or very low numbers of reads*  *-- Consider your complete workload, and how long your instance has been running*  *-- Investigate further before dropping an index!*  *-- Missing Indexes for current database by Index Advantage (Query 70) (Missing Indexes)*  **SELECT** CONVERT(**decimal**(18,2), migs.user\_seeks \* migs.avg\_total\_user\_cost \* (migs.avg\_user\_impact \* 0.01)) **AS** [index\_advantage],  CONVERT(**nvarchar**(25), migs.last\_user\_seek, 20) **AS** [last\_user\_seek],  mid.[statement] **AS** [Database.Schema.Table],  COUNT(1) **OVER**(**PARTITION** **BY** mid.[statement]) **AS** [missing\_indexes\_for\_table],  COUNT(1) **OVER**(**PARTITION** **BY** mid.[statement], mid.equality\_columns) **AS** [similar\_missing\_indexes\_for\_table],  mid.equality\_columns, mid.inequality\_columns, mid.included\_columns, migs.user\_seeks,  CONVERT(**decimal**(18,2), migs.avg\_total\_user\_cost) **AS** [avg\_total\_user\_,cost], migs.avg\_user\_impact,  REPLACE(REPLACE(LEFT(st.[text], 255), **CHAR**(10),''), **CHAR**(13),'') **AS** [Short Query Text],  OBJECT\_NAME(mid.[object\_id]) **AS** [Table Name], p.**rows** **AS** [Table Rows]  **FROM** sys.dm\_db\_missing\_index\_groups **AS** mig **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_db\_missing\_index\_group\_stats\_query **AS** migs **WITH**(NOLOCK)  **ON** mig.index\_group\_handle = migs.group\_handle  **CROSS** APPLY sys.dm\_exec\_sql\_text(migs.last\_sql\_handle) **AS** st  **INNER** **JOIN** sys.dm\_db\_missing\_index\_details **AS** mid **WITH** (NOLOCK)  **ON** mig.index\_handle = mid.index\_handle  **INNER** **JOIN** sys.partitions **AS** p **WITH** (NOLOCK)  **ON** p.[object\_id] = mid.[object\_id]  **WHERE** mid.database\_id = DB\_ID()  **AND** p.index\_id < 2  **ORDER** **BY** index\_advantage **DESC** **OPTION** (RECOMPILE);  *------*  *-- Look at index advantage, last user seek time, number of user seeks to help determine source and importance*  *-- SQL Server is overly eager to add included columns, so beware*  *-- Do not just blindly add indexes that show up from this query!!!*  *-- Hkan Winther has given me some great suggestions for this query*  *-- Find missing index warnings for cached plans in the current database (Query 71) (Missing Index Warnings)*  *-- Note: This query could take some time on a busy instance*  **SELECT** **TOP**(25) OBJECT\_NAME(objectid) **AS** [ObjectName],  cp.objtype, cp.usecounts, cp.size\_in\_bytes, qp.query\_plan  **FROM** sys.dm\_exec\_cached\_plans **AS** cp **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_query\_plan(cp.plan\_handle) **AS** qp  **WHERE** CAST(qp.query\_plan **AS** **NVARCHAR**(MAX)) **LIKE** N'%MissingIndex%'  **AND** qp.dbid = DB\_ID()  **ORDER** **BY** cp.usecounts **DESC** **OPTION** (RECOMPILE);  *------*  *-- Helps you connect missing indexes to specific stored procedures or queries*  *-- This can help you decide whether to add them or not*  *-- Breaks down buffers used by current database by object (table, index) in the buffer cache (Query 72) (Buffer Usage)*  *-- Note: This query could take some time on a busy instance*  **SELECT** fg.name **AS** [Filegroup Name], SCHEMA\_NAME(o.schema\_id) **AS** [Schema Name],  OBJECT\_NAME(p.[object\_id]) **AS** [Object Name], p.index\_id,  CAST(COUNT(\*)/128.0 **AS** **DECIMAL**(10, 2)) **AS** [Buffer size(MB)],  COUNT(\*) **AS** [BufferCount], p.[rows] **AS** [Row Count],  p.data\_compression\_desc **AS** [Compression Type]  **FROM** sys.allocation\_units **AS** a **WITH** (NOLOCK)  **INNER** **JOIN** sys.dm\_os\_buffer\_descriptors **AS** b **WITH** (NOLOCK)  **ON** a.allocation\_unit\_id = b.allocation\_unit\_id  **INNER** **JOIN** sys.partitions **AS** p **WITH** (NOLOCK)  **ON** a.container\_id = p.hobt\_id  **INNER** **JOIN** sys.objects **AS** o **WITH** (NOLOCK)  **ON** p.object\_id = o.object\_id  **INNER** **JOIN** sys.database\_files **AS** f **WITH** (NOLOCK)  **ON** b.file\_id = f.file\_id  **INNER** **JOIN** sys.filegroups **AS** fg **WITH** (NOLOCK)  **ON** f.data\_space\_id = fg.data\_space\_id  **WHERE** b.database\_id = CONVERT(**int**, DB\_ID())  **AND** p.[object\_id] > 100  **AND** OBJECT\_NAME(p.[object\_id]) **NOT** **LIKE** N'plan\_%'  **AND** OBJECT\_NAME(p.[object\_id]) **NOT** **LIKE** N'sys%'  **AND** OBJECT\_NAME(p.[object\_id]) **NOT** **LIKE** N'xml\_index\_nodes%'  **GROUP** **BY** fg.name, o.schema\_id, p.[object\_id], p.index\_id,  p.data\_compression\_desc, p.[rows]  **ORDER** **BY** [BufferCount] **DESC** **OPTION** (RECOMPILE);  *------*  *-- Tells you what tables and indexes are using the most memory in the buffer cache*  *-- It can help identify possible candidates for data compression*  *-- Get Schema names, Table names, object size, row counts, and compression status for clustered index or heap (Query 73) (Table Sizes)*  **SELECT** DB\_NAME(DB\_ID()) **AS** [Database Name], SCHEMA\_NAME(o.schema\_id) **AS** [Schema Name],  OBJECT\_NAME(p.object\_id) **AS** [Table Name],  CAST(SUM(ps.reserved\_page\_count) \* 8.0 / 1024 **AS** **DECIMAL**(19,2)) **AS** [Object Size (MB)],  SUM(p.**rows**) **AS** [Row Count],  p.data\_compression\_desc **AS** [Compression Type]  **FROM** sys.objects **AS** o **WITH** (NOLOCK)  **INNER** **JOIN** sys.partitions **AS** p **WITH** (NOLOCK)  **ON** p.object\_id = o.object\_id  **INNER** **JOIN** sys.dm\_db\_partition\_stats **AS** ps **WITH** (NOLOCK)  **ON** p.object\_id = ps.object\_id  **WHERE** ps.index\_id < 2 *-- ignore the partitions from the non-clustered indexes if any*  **AND** p.index\_id < 2 *-- ignore the partitions from the non-clustered indexes if any*  **AND** o.type\_desc = N'USER\_TABLE'  **GROUP** **BY** SCHEMA\_NAME(o.schema\_id), p.object\_id, ps.reserved\_page\_count, p.data\_compression\_desc  **ORDER** **BY** SUM(ps.reserved\_page\_count) **DESC**, SUM(p.**rows**) **DESC** **OPTION** (RECOMPILE);  *------*  *-- Gives you an idea of table sizes, and possible data compression opportunities*  *-- Get some key table properties (Query 74) (Table Properties)*  **SELECT** OBJECT\_NAME(t.[object\_id]) **AS** [ObjectName], p.[rows] **AS** [Table Rows], p.index\_id,  p.data\_compression\_desc **AS** [Index Data Compression],  t.create\_date, t.lock\_on\_bulk\_load, t.is\_replicated, t.has\_replication\_filter,  t.is\_tracked\_by\_cdc, t.lock\_escalation\_desc, t.is\_filetable,  t.is\_memory\_optimized, t.durability\_desc,  t.temporal\_type\_desc, t.is\_remote\_data\_archive\_enabled, t.is\_external  **FROM** sys.tables **AS** t **WITH** (NOLOCK)  **INNER** **JOIN** sys.partitions **AS** p **WITH** (NOLOCK)  **ON** t.[object\_id] = p.[object\_id]  **WHERE** OBJECT\_NAME(t.[object\_id]) **NOT** **LIKE** N'sys%'  **ORDER** **BY** OBJECT\_NAME(t.[object\_id]), p.index\_id **OPTION** (RECOMPILE);  *------*  *-- Gives you some good information about your tables*  *-- is\_memory\_optimized and durability\_desc were new in SQL Server 2014*  *-- temporal\_type\_desc, is\_remote\_data\_archive\_enabled, is\_external were new in SQL Server 2016*  *-- sys.tables (Transact-SQL)*  *-- https://bit.ly/2Gk7998*  *-- When were Statistics last updated on all indexes? (Query 75) (Statistics Update)*  **SELECT** SCHEMA\_NAME(o.schema\_id) + N'.' + o.[name] **AS** [Object Name], o.[type\_desc] **AS** [Object Type],  i.[name] **AS** [Index Name], STATS\_DATE(i.[object\_id], i.index\_id) **AS** [Statistics Date],  s.auto\_created, s.no\_recompute, s.user\_created, s.is\_incremental, s.is\_temporary,  s.has\_persisted\_sample, sp.persisted\_sample\_percent,  (sp.rows\_sampled \* 100)/sp.**rows** **AS** [Actual Sample Percent], sp.modification\_counter,  st.row\_count, st.used\_page\_count  **FROM** sys.objects **AS** o **WITH** (NOLOCK)  **INNER** **JOIN** sys.indexes **AS** i **WITH** (NOLOCK)  **ON** o.[object\_id] = i.[object\_id]  **INNER** **JOIN** sys.stats **AS** s **WITH** (NOLOCK)  **ON** i.[object\_id] = s.[object\_id]  **AND** i.index\_id = s.stats\_id  **INNER** **JOIN** sys.dm\_db\_partition\_stats **AS** st **WITH** (NOLOCK)  **ON** o.[object\_id] = st.[object\_id]  **AND** i.[index\_id] = st.[index\_id]  **CROSS** APPLY sys.dm\_db\_stats\_properties(s.object\_id, s.stats\_id) **AS** sp  **WHERE** o.[type] **IN** ('U', 'V')  **AND** st.row\_count > 0  **ORDER** **BY** STATS\_DATE(i.[object\_id], i.index\_id) **DESC** **OPTION** (RECOMPILE);  *------*  *-- Helps discover possible problems with out-of-date statistics*  *-- Also gives you an idea which indexes are the most active*  *-- sys.stats (Transact-SQL)*  *-- https://bit.ly/2GyAxrn*  *-- UPDATEs to Statistics (Erin Stellato)*  *-- https://bit.ly/2vhrYQy*  *-- Look at most frequently modified indexes and statistics (Query 76) (Volatile Indexes)*  **SELECT** o.[name] **AS** [Object Name], o.[object\_id], o.[type\_desc], s.[name] **AS** [Statistics Name],  s.stats\_id, s.no\_recompute, s.auto\_created, s.is\_incremental, s.is\_temporary,  sp.modification\_counter, sp.[rows], sp.rows\_sampled, sp.last\_updated  **FROM** sys.objects **AS** o **WITH** (NOLOCK)  **INNER** **JOIN** sys.stats **AS** s **WITH** (NOLOCK)  **ON** s.object\_id = o.object\_id  **CROSS** APPLY sys.dm\_db\_stats\_properties(s.object\_id, s.stats\_id) **AS** sp  **WHERE** o.[type\_desc] **NOT** **IN** (N'SYSTEM\_TABLE', N'INTERNAL\_TABLE')  **AND** sp.modification\_counter > 0  **ORDER** **BY** sp.modification\_counter **DESC**, o.name **OPTION** (RECOMPILE);  *------*  *-- This helps you understand your workload and make better decisions about*  *-- things like data compression and adding new indexes to a table*  *-- Get fragmentation info for all indexes above a certain size in the current database (Query 77) (Index Fragmentation)*  *-- Note: This query could take some time on a very large database*  **SELECT** DB\_NAME(ps.database\_id) **AS** [Database Name], SCHEMA\_NAME(o.[schema\_id]) **AS** [Schema Name],  OBJECT\_NAME(ps.object\_id) **AS** [Object Name], i.[name] **AS** [Index Name], ps.index\_id, ps.index\_type\_desc,  CAST(ps.avg\_fragmentation\_in\_percent **AS** **DECIMAL** (15,3)) **AS** [Avg Fragmentation in Pct],  ps.fragment\_count, ps.page\_count, i.fill\_factor, i.has\_filter, i.filter\_definition, i.[allow\_page\_locks]  **FROM** sys.dm\_db\_index\_physical\_stats(DB\_ID(),**NULL**, **NULL**, **NULL** , N'LIMITED') **AS** ps  **INNER** **JOIN** sys.indexes **AS** i **WITH** (NOLOCK)  **ON** ps.[object\_id] = i.[object\_id]  **AND** ps.index\_id = i.index\_id  **INNER** **JOIN** sys.objects **AS** o **WITH** (NOLOCK)  **ON** i.[object\_id] = o.[object\_id]  **WHERE** ps.database\_id = DB\_ID()  **AND** ps.page\_count > 2500  **ORDER** **BY** ps.avg\_fragmentation\_in\_percent **DESC** **OPTION** (RECOMPILE);  *------*  *-- Helps determine whether you have framentation in your relational indexes*  *-- and how effective your index maintenance strategy is*  *--- Index Read/Write stats (all tables in current DB) ordered by Reads (Query 78) (Overall Index Usage - Reads)*  **SELECT** SCHEMA\_NAME(t.[schema\_id]) **AS** [SchemaName], OBJECT\_NAME(i.[object\_id]) **AS** [ObjectName],  i.[name] **AS** [IndexName], i.index\_id, i.[type\_desc] **AS** [Index Type],  s.user\_seeks, s.user\_scans, s.user\_lookups,  s.user\_seeks + s.user\_scans + s.user\_lookups **AS** [Total Reads],  s.user\_updates **AS** [Writes],  i.fill\_factor **AS** [Fill Factor], i.has\_filter, i.filter\_definition,  s.last\_user\_scan, s.last\_user\_lookup, s.last\_user\_seek, i.[allow\_page\_locks], i.[allow\_row\_locks],  i.[optimize\_for\_sequential\_key]  **FROM** sys.indexes **AS** i **WITH** (NOLOCK)  LEFT **OUTER** **JOIN** sys.dm\_db\_index\_usage\_stats **AS** s **WITH** (NOLOCK)  **ON** i.[object\_id] = s.[object\_id]  **AND** i.index\_id = s.index\_id  **AND** s.database\_id = DB\_ID()  LEFT **OUTER** **JOIN** sys.tables **AS** t **WITH** (NOLOCK)  **ON** t.[object\_id] = i.[object\_id]  **WHERE** OBJECTPROPERTY(i.[object\_id],'IsUserTable') = 1  **ORDER** **BY** s.user\_seeks + s.user\_scans + s.user\_lookups **DESC** **OPTION** (RECOMPILE); *-- Order by reads*  *------*  *-- Show which indexes in the current database are most active for Reads*  *--- Index Read/Write stats (all tables in current DB) ordered by Writes (Query 79) (Overall Index Usage - Writes)*  **SELECT** SCHEMA\_NAME(t.[schema\_id]) **AS** [SchemaName],OBJECT\_NAME(i.[object\_id]) **AS** [ObjectName],  i.[name] **AS** [IndexName], i.index\_id, i.[type\_desc] **AS** [Index Type],  s.user\_updates **AS** [Writes], s.user\_seeks + s.user\_scans + s.user\_lookups **AS** [Total Reads],  i.fill\_factor **AS** [Fill Factor], i.has\_filter, i.filter\_definition,  s.last\_system\_update, s.last\_user\_update, i.[allow\_page\_locks], i.[allow\_row\_locks],  i.[optimize\_for\_sequential\_key]  **FROM** sys.indexes **AS** i **WITH** (NOLOCK)  LEFT **OUTER** **JOIN** sys.dm\_db\_index\_usage\_stats **AS** s **WITH** (NOLOCK)  **ON** i.[object\_id] = s.[object\_id]  **AND** i.index\_id = s.index\_id  **AND** s.database\_id = DB\_ID()  LEFT **OUTER** **JOIN** sys.tables **AS** t **WITH** (NOLOCK)  **ON** t.[object\_id] = i.[object\_id]  **WHERE** OBJECTPROPERTY(i.[object\_id],'IsUserTable') = 1  **ORDER** **BY** s.user\_updates **DESC** **OPTION** (RECOMPILE); *-- Order by writes*  *------*  *-- Show which indexes in the current database are most active for Writes*  *-- Get lock waits for current database (Query 80) (Lock Waits)*  **SELECT** o.name **AS** [table\_name], i.name **AS** [index\_name], ios.index\_id, ios.partition\_number,  SUM(ios.row\_lock\_wait\_count) **AS** [total\_row\_lock\_waits],  SUM(ios.row\_lock\_wait\_in\_ms) **AS** [total\_row\_lock\_wait\_in\_ms],  SUM(ios.index\_lock\_promotion\_attempt\_count) **AS** [total index\_lock\_promotion\_attempt\_count],  SUM(ios.index\_lock\_promotion\_count) **AS** [ios.index\_lock\_promotion\_count],  SUM(ios.page\_lock\_wait\_count) **AS** [total\_page\_lock\_waits],  SUM(ios.page\_lock\_wait\_in\_ms) **AS** [total\_page\_lock\_wait\_in\_ms],  SUM(ios.page\_lock\_wait\_in\_ms)+ SUM(row\_lock\_wait\_in\_ms) **AS** [total\_lock\_wait\_in\_ms]  **FROM** sys.dm\_db\_index\_operational\_stats(DB\_ID(), **NULL**, **NULL**, **NULL**) **AS** ios  **INNER** **JOIN** sys.objects **AS** o **WITH** (NOLOCK)  **ON** ios.[object\_id] = o.[object\_id]  **INNER** **JOIN** sys.indexes **AS** i **WITH** (NOLOCK)  **ON** ios.[object\_id] = i.[object\_id]  **AND** ios.index\_id = i.index\_id  **WHERE** o.[object\_id] > 100  **GROUP** **BY** o.name, i.name, ios.index\_id, ios.partition\_number  **HAVING** SUM(ios.page\_lock\_wait\_in\_ms)+ SUM(row\_lock\_wait\_in\_ms) > 0  **ORDER** **BY** total\_lock\_wait\_in\_ms **DESC** **OPTION** (RECOMPILE);  *------*  *-- This query is helpful for troubleshooting blocking and deadlocking issues*  *-- sys.dm\_db\_index\_operational\_stats (Transact-SQL)*  *-- https://bit.ly/3l5rGEw*  *-- Look at UDF execution statistics (Query 81) (UDF Statistics)*  **SELECT** OBJECT\_NAME(object\_id) **AS** [Function Name], execution\_count,  total\_worker\_time, total\_worker\_time/execution\_count **AS** [avg\_worker\_time],  total\_logical\_reads, total\_physical\_reads, total\_elapsed\_time,  total\_elapsed\_time/execution\_count **AS** [avg\_elapsed\_time],  CONVERT(**nvarchar**(25), last\_execution\_time, 20) **AS** [Last Execution Time],  CONVERT(**nvarchar**(25), cached\_time, 20) **AS** [Plan Cached Time]  **FROM** sys.dm\_exec\_function\_stats **WITH** (NOLOCK)  **WHERE** database\_id = DB\_ID()  **ORDER** **BY** total\_worker\_time **DESC** **OPTION** (RECOMPILE);  *------*  *-- New for SQL Server 2016*  *-- Helps you investigate scalar UDF performance issues*  *-- Does not return information for table valued functions*  *-- sys.dm\_exec\_function\_stats (Transact-SQL)*  *-- https://bit.ly/2q1Q6BM*  *-- Determine which scalar UDFs are in-lineable (Query 82) (Inlineable UDFs)*  **SELECT** OBJECT\_NAME(m.object\_id) **AS** [Function Name], is\_inlineable, inline\_type,  efs.total\_worker\_time  **FROM** sys.sql\_modules **AS** m **WITH** (NOLOCK)  LEFT **OUTER** **JOIN** sys.dm\_exec\_function\_stats **AS** efs **WITH** (NOLOCK)  **ON** m.object\_id = efs.object\_id  **WHERE** efs.type\_desc = N'SQL\_SCALAR\_FUNCTION'  **ORDER** **BY** efs.total\_worker\_time **DESC**  **OPTION** (RECOMPILE);  *------*  *-- Scalar UDF Inlining*  *-- https://bit.ly/2JU971M*  *-- sys.sql\_modules (Transact-SQL)*  *-- https://bit.ly/2Qt216S*  *-- Get Query Store Options for this database (Query 83) (Query Store Options)*  **SELECT** actual\_state\_desc, desired\_state\_desc, [interval\_length\_minutes],  current\_storage\_size\_mb, [max\_storage\_size\_mb],  query\_capture\_mode\_desc, size\_based\_cleanup\_mode\_desc, wait\_stats\_capture\_mode\_desc  **FROM** sys.database\_query\_store\_options **WITH** (NOLOCK) **OPTION** (RECOMPILE);  *------*  *-- New for SQL Server 2016*  *-- Requires that Query Store is enabled for this database*  *-- Make sure that the actual\_state\_desc is the same as desired\_state\_desc*  *-- Make sure that the current\_storage\_size\_mb is less than the max\_storage\_size\_mb*  *-- Tuning Workload Performance with Query Store*  *-- https://bit.ly/1kHSl7w*  *-- Emergency shutoff for Query Store (SQL Server 2019 CU6 or newer)*  *-- ALTER DATABASE [DatabaseName] SET QUERY\_STORE = OFF(FORCED);*  *-- Get input buffer information for the current database (Query 84) (Input Buffer)*  **SELECT** es.session\_id, DB\_NAME(es.database\_id) **AS** [Database Name],  es.[program\_name], es.[host\_name], es.login\_name,  es.login\_time, es.cpu\_time, es.logical\_reads, es.memory\_usage,  es.[status], ib.event\_info **AS** [Input Buffer]  **FROM** sys.dm\_exec\_sessions **AS** es **WITH** (NOLOCK)  **CROSS** APPLY sys.dm\_exec\_input\_buffer(es.session\_id, **NULL**) **AS** ib  **WHERE** es.database\_id = DB\_ID()  **AND** es.session\_id > 50  **AND** es.session\_id <> @@SPID **OPTION** (RECOMPILE);  *------*  *-- Gives you input buffer information from all non-system sessions for the current database*  *-- Replaces DBCC INPUTBUFFER*  *-- New DMF for retrieving input buffer in SQL Server*  *-- https://bit.ly/2uHKMbz*  *-- sys.dm\_exec\_input\_buffer (Transact-SQL)*  *-- https://bit.ly/2J5Hf9q*  *-- Get any resumable index rebuild operation information (Query 85) (Resumable Index Rebuild)*  **SELECT** OBJECT\_NAME(iro.object\_id) **AS** [Object Name], iro.index\_id, iro.name **AS** [Index Name],  iro.sql\_text, iro.last\_max\_dop\_used, iro.partition\_number, iro.state\_desc,  iro.start\_time, CONVERT(**decimal**(15,2),iro.percent\_complete) **AS** [Percent Complete],  iro.last\_pause\_time, iro.total\_execution\_time **AS** [Execution Min],  CONVERT(**decimal**(15,2),iro.total\_execution\_time \* (100.0 - iro.percent\_complete)/iro.percent\_complete) **AS** [Approx Execution Min Left]  **FROM** sys.index\_resumable\_operations **AS** iro **WITH** (NOLOCK)  **OPTION** (RECOMPILE);  *------*  *-- index\_resumable\_operations (Transact-SQL)*  *-- https://bit.ly/2pYSWqq*  *-- Get database automatic tuning options (Query 86) (Automatic Tuning Options)*  **SELECT** [name], desired\_state\_desc, actual\_state\_desc, reason\_desc  **FROM** sys.database\_automatic\_tuning\_options **WITH** (NOLOCK)  **OPTION** (RECOMPILE);  *------*  *-- sys.database\_automatic\_tuning\_options (Transact-SQL)*  *-- https://bit.ly/2FHhLkL*  *-- Look at recent Full backups for the current database (Query 87) (Recent Full Backups)*  **SELECT** **TOP** (30) bs.machine\_name, bs.server\_name, bs.database\_name **AS** [Database Name], bs.recovery\_model,  CONVERT (**BIGINT**, bs.backup\_size / 1048576 ) **AS** [Uncompressed Backup Size (MB)],  CONVERT (**BIGINT**, bs.compressed\_backup\_size / 1048576 ) **AS** [Compressed Backup Size (MB)],  CONVERT (**NUMERIC** (20,2), (CONVERT (**FLOAT**, bs.backup\_size) /  CONVERT (**FLOAT**, bs.compressed\_backup\_size))) **AS** [Compression Ratio], bs.has\_backup\_checksums, bs.is\_copy\_only, bs.encryptor\_type,  DATEDIFF (**SECOND**, bs.backup\_start\_date, bs.backup\_finish\_date) **AS** [Backup Elapsed Time (sec)],  bs.backup\_finish\_date **AS** [Backup Finish Date], bmf.physical\_device\_name **AS** [Backup Location], bmf.physical\_block\_size  **FROM** msdb.dbo.backupset **AS** bs **WITH** (NOLOCK)  **INNER** **JOIN** msdb.dbo.backupmediafamily **AS** bmf **WITH** (NOLOCK)  **ON** bs.media\_set\_id = bmf.media\_set\_id  **WHERE** bs.database\_name = DB\_NAME(DB\_ID())  **AND** bs.[type] = 'D' *-- Change to L if you want Log backups*  **ORDER** **BY** bs.backup\_finish\_date **DESC** **OPTION** (RECOMPILE);  *------*  *-- Things to look at:*  *-- Are your backup sizes and times changing over time?*  *-- Are you using backup compression?*  *-- Are you using backup checksums?*  *-- Are you doing copy\_only backups?*  *-- Are you doing encrypted backups?*  *-- Have you done any backup tuning with striped backups, or changing the parameters of the backup command?*  *-- Where are the backups going to?*  *-- In SQL Server 2016 and newer, native SQL Server backup compression actually works*  *-- much better with databases that are using TDE than in previous versions*  *-- https://bit.ly/28Rpb2x*  *-- Microsoft Visual Studio Dev Essentials*  *-- https://bit.ly/2qjNRxi*  *-- Microsoft Azure Learn*  *-- https://bit.ly/2O0Hacc* |