Knose Database Monitoring Utility

Introduction

The Knose *(KNOwledgeable SQL Extraction)* tools have been developed by VerifAction for logging all database queries, enabling full auditing of application database queries. Knose facilitates fast problem diagnosis such as uncovering long running and inefficient queries, as well as providing a reliable means for general system health monitoring. Accidental or malicious damage to a database is traceable providing the identification of the perpetrator and the sequence of commands surrounding the incident. But more specifically, the utility facilitates complex problem resolution within an application, allowing site-specific findings to be clearly communicated to the DBAs, the application vendors and the developers.

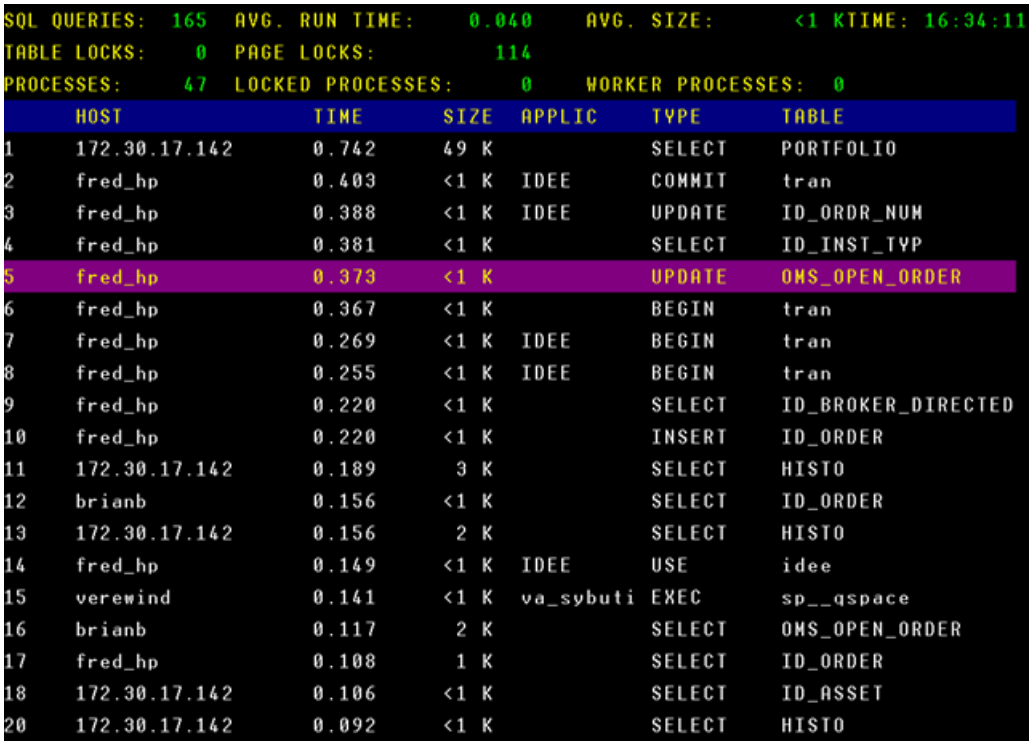
Knose extracts data from a low-level network layer and uses a reverse TCP stack to reconstruct the session back to the application (database) layer. This technique allows monitoring and timestamping of all queries and data entering and leaving the database server, ensuring an unbiased view of system performance. In addition, through timestamping the data returning to the client Knose can accurately determine the performance of each client as well as the efficiency of the network.

Knose includes a proxy module for certain operating systems (eg Solaris) in which the local traffic bypasses the low level IP stack. The proxy monitors internal processing (batch jobs etc) which are initiated on the server itself.

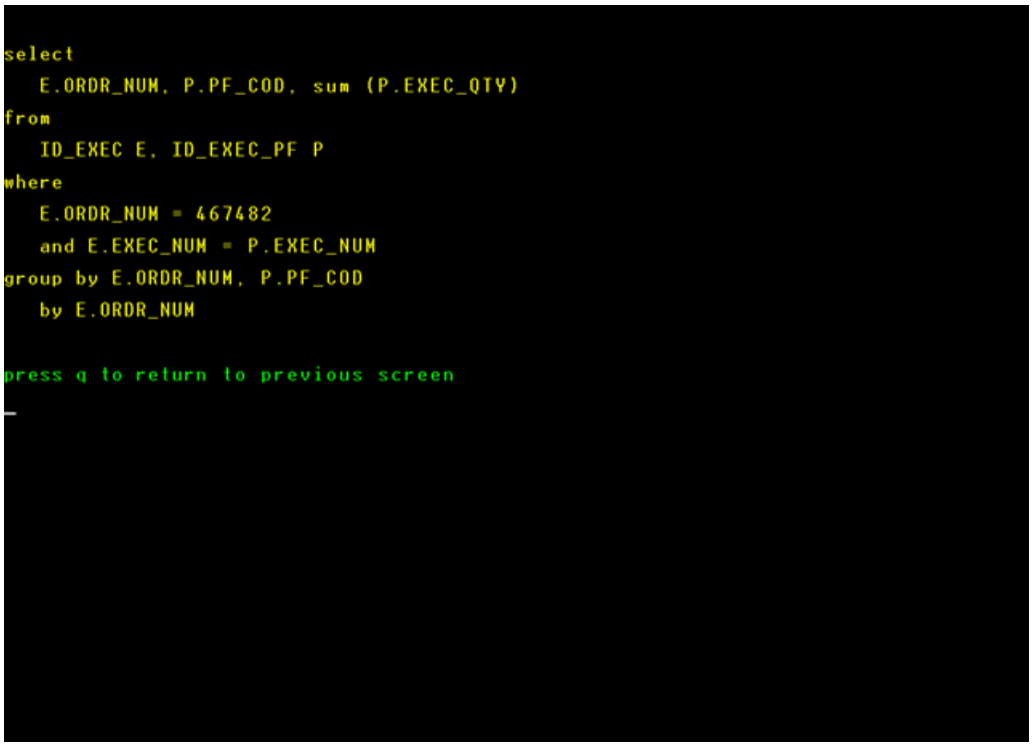
Knose, has been ported to Sybase, Oracle and MicroSoft SQL databases.

WhoKnose console

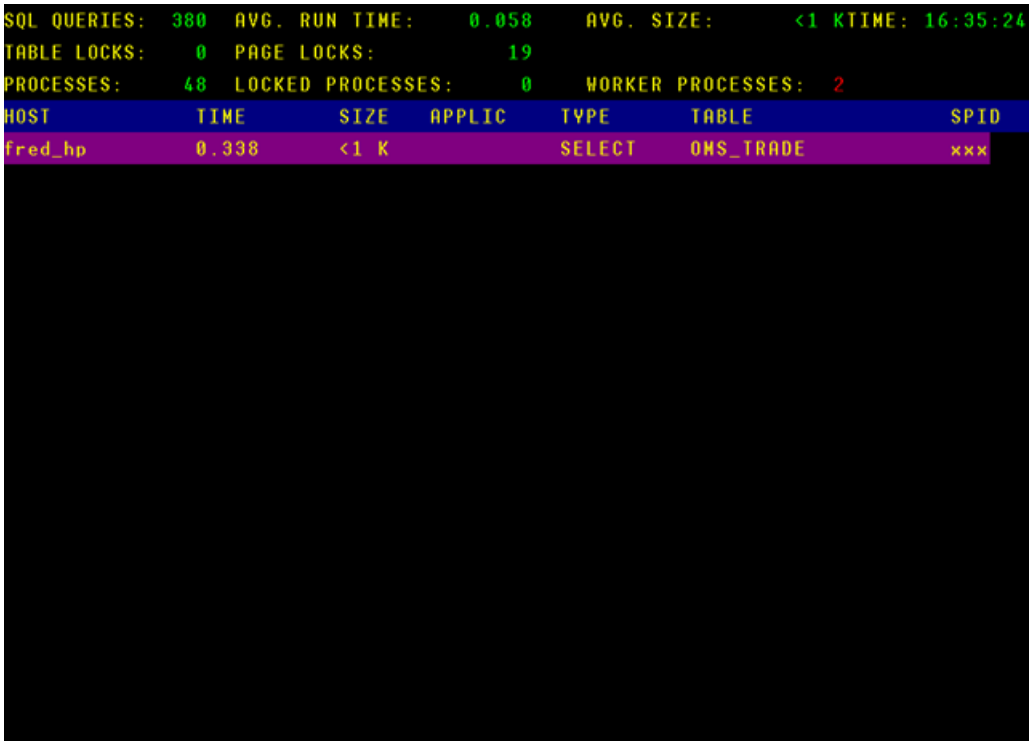
WhoKnose *(pr "hooknose")* is used to clearly present sql performance information on a multi-user, live system. A number of displays provide verbose and summary information: The detailed query screen (below) shows all queries that have run in the last few minutes. The display can be sorted by run-time, execution time, application IP adress etc.



Pressing enter on a specific line item displays the complete listing of the query:



Another useful screen is that of currently running queries. Whereas the previous screens showed queries that had recently completed, this screen shows queries that are currently being processed by the database engines. Again, pressing enter on a specific line item displays the complete listing of the query.



WhoKnose also includes a screen for monitoring database locks and blocks.

Knose Logging

Knose stores data in a set of text based log files, rolling the files over on a daily basis. Every SQL query is logged by time-stamp, the IP address where the query originated and the amount of data returned.

Without complicated analysis, these log files are very useful for many auditing purposes. It is a simple matter to search through the logs to see when a particular query was run. For example, if a table has been dropped, or a stored procedure has been rolled in, the log files will show which user ran the query and when it was submitted.

Locating performance problems

In its simplest form performance analysis involves scanning the sql logs to analyse the completed queries and summarising its frequency and run-time. The Knose ***sqlanalyze*** utility provides an easy way of doing this. One method it uses is to simplify the SQL statements, remove all references to specific dates, names values etc, leaving just the bare SQL commands. To generate a list of the longest running queries, for instance, use the following command:

```
sqlanalyze -v -t -minsql 1 sql_200308*.txt > allsql.txt
```

This returns:

```
  TOT-SQL     COUNT        SQL         PC  nSQL     CRC     DETAIL
1798.996         2    899.498      0.000     1 0015499c8bd SELECT pt_trmast.tmeffdate,pt_trmast.tm
 439.306         5     87.861      0.000     1 0015cc49798 SELECT rdm_portfolio_dim.pdr_port_name,
 276.959      1963      0.141      0.000     1 0016bf1b6bb SELECT oir_batch.btc_batch_id,oir_batch
 218.154         1    218.154      0.000     1 0017aedccce SELECT '?',pr_secname.smdesc1 '?',prntn
 179.865         1    179.865      0.000     1 0010a75c7ec SELECT '?',pr_secname.smdesc1 '?',prntn
 143.097         1    143.097      0.000     1 0015aff48ca SELECT dbo.vwAMBondYield.pri_yield,dbo.
 140.078         3     46.693      0.000     1 0010ded43bd DELETE FROM ms_user_par_restr WHERE NOT
  69.509         3     23.170      0.000     1 001fe65ee41 SELECT(SELECT pr_secname.smdesc1 FROM p
```

```
    49.993       18      2.777      0.000      1 0017323a12e EXECUTE gpssp_OpenOrders;1 @fundmanager
    46.068        8      5.759      0.000      1 00131b3c78a SELECT pr_account_view.acacname,fot_tra
    28.624        4      7.156      0.000      1 001e200460c ortfolio_dim.pdr_port_name,vwAMBondYiel
    24.863       12      2.072      0.000      1 001c61dbe7b SELECT rt.phperiod,rt.phrectype,rt.phmr
    18.637       28      0.666      0.000      1 00175702d19 SELECT c.usertype,o.id,o.name,c.id,c.na
    17.477       12      1.456      0.000      1 0016c8b905a SELECT rdm_trans_fact.tfr_trans_categor
    16.838        3      5.613      0.000      1 001e67a5ff4 INSERT INTO ms_user_par_restr SELECT DI
    14.800        3      4.933      0.000      1 001e947ea53 trans_fact.tfr_trans_category,hilive.db
    14.492        1     14.492      0.000      1 001f35abca5 SELECT rdm_valuation_fact.vfr_book_cost
    13.257        6      2.210      0.000      1 001c58182b3 SELECT pr_account_view.acacname,fot_tra
    12.926        8      1.616      0.000      1 0011d48078f SELECT rdm_portfolio_dim.pdr_port_name,
    12.437      354      0.035      0.000      1 00102c584f4 COMMIT TRAN
    12.399        1     12.399      0.000      1 001c354eb0b SELECT hilive.dbo.vwSecNameL5.L5Name,hi
```

The full (long) SQL statement is extracted but as it is kept to a single line for ease of processing, the SQL beautifier switch can used to present the SQL in a readable format. Before beautifying, the first query above looks like this:

```
SELECT pt_trmast.tmeffdate,pt_trmast.tmtradedate,pt_trmast.txtmid,pt_trmast.tmcode,pr_secname.smshor
tdesc,pr_currency.currcode,pt_trmast.tmpricesetl,pt_trmast.tmsetdate,pr_trancode.longshort,pr_transd
at.buysell,pt_trmast.tmquantity,pr_account_view.acnominor,pr_account_view.acacname,space(#)AS gps_ty
pes,pt_trmast.setlamtsetl,pt_trmast.tm_price_ccy_id FROM pt_trmast,pr_trancode,pr_currency,pt_trlink
,pr_tradestage,pr_transdat,pr_account_view,pr_secname WHERE(pt_trmast.acid *=pt_trlink.acid1)AND(pt_
trmast.tmid *=pt_trlink.tmid1)AND(pt_trmast.tmcode *=pr_transdat.tmcode)AND(pt_trmast.tmcode=pr_tran
code.trancode)AND(pt_trmast.tm_price_ccy_id=pr_currency.smsecid)AND(pt_trmast.tmcode=pr_tradestage.t
rd_tmcode)AND(pt_trmast.acid=pr_account_view.acid)AND(pt_trmast.tmsecid=pr_secname.smsecid)AND(pt_tr
mast.acid IN(#))AND(pt_trmast.tm_setlbac_secid1=#)AND(pt_trlink.tlreason='?' OR pt_trlink.tlreason='
?')AND(pr_secname.smdate<='?' AND pr_secname.smenddate>'?')AND(pr_tradestage.trd_trdstg IN(#,#))AND(
pt_trmast.tmeffdate>='?' AND pt_trmast.tmeffdate<='?')AND((#=#)OR(#=# AND pt_trmast.cancel_date='?')
)AND NOT EXISTS(SELECT * FROM pt_trmast tr2 WHERE tr2.tm_expirydate<>'?' AND tr2.tmtradedate<='?' AN
D tr2.txtmid=pt_trmast.txtmid)ORDER BY pt_trmast.tmeffdate DESC
```

Using the –sql switch:

```
sqlanalyze -v -sql -crc 0015499c8bd sql_200308*.txt

-------------------------------------------------------------------------------
     SELECT
        pt_trmast.tmeffdate, pt_trmast.tmtradedate, pt_trmast.txtmid,
        pt_trmast.tmcode, pr_secname.smshortdesc, pr_currency.currcode,
        pt_trmast.tmpricesetl, pt_trmast.tmsetdate, pr_trancode.longshort,
        pr_transdat.buysell, pt_trmast.tmquantity, pr_account_view.acnominor,
        pr_account_view.acacname, space (#) AS gps_types, pt_trmast.setlamtsetl,
        pt_trmast.tm_price_ccy_id
     FROM
        pt_trmast, pr_trancode, pr_currency, pt_trlink, pr_tradestage, pr_transdat,
        pr_account_view, pr_secname
     WHERE
        (pt_trmast.acid * = pt_trlink.acid1)
        AND (pt_trmast.tmid * = pt_trlink.tmid1)
        AND (pt_trmast.tmcode * = pr_transdat.tmcode)
        AND (pt_trmast.tmcode = pr_trancode.trancode)
        AND (pt_trmast.tm_price_ccy_id = pr_currency.smsecid)
        AND (pt_trmast.tmcode = pr_tradestage.trd_tmcode)
        AND (pt_trmast.acid = pr_account_view.acid)
        AND (pt_trmast.tmsecid = pr_secname.smsecid)
        AND (pt_trmast.acid IN (#))
        AND (pt_trmast.tm_setlbac_secid1 = #)
        AND (pt_trlink.tlreason = '?'
        OR pt_trlink.tlreason = '?')
        AND (pr_secname.smdate <= '?'
        AND pr_secname.smenddate > '?')
        AND (pr_tradestage.trd_trdstg IN (#, #))
        AND (pt_trmast.tmeffdate >= '?'
        AND pt_trmast.tmeffdate <= '?')
        AND ((# = #)
        OR (# = #
        AND pt_trmast.cancel_date = '?'))
        AND NOT EXISTS (
     SELECT
        *
     FROM
        pt_trmast tr2
     WHERE
```

```
      tr2.tm_expirydate < > '?'
      AND tr2.tmtradedate <= '?'
      AND tr2.txtmid = pt_trmast.txtmid)
   ORDER BY pt_trmast.tmeffdate DESC
```

The –sample switch, on the otherhand, allows the extraction of the raw (unstripped) query, with all the values clearly visible:

```
sqlanalyze -v -sample 1 -crc 0015499c8bd sql_200308*.txt

--------------------------------------------------------------------------------
2003/08/01 08:54:09.872  1542.196 seconds

   SELECT
      pt_trmast.tmeffdate, pt_trmast.tmtradedate, pt_trmast.txtmid,
      pt_trmast.tmcode, pr_secname.smshortdesc, pr_currency.currcode,
      pt_trmast.tmpricesetl, pt_trmast.tmsetdate, pr_trancode.longshort,
      pr_transdat.buysell, pt_trmast.tmquantity, pr_account_view.acnominor,
      pr_account_view.acacname, space (4) as gps_types, pt_trmast.setlamtsetl,
      pt_trmast.tm_price_ccy_id
   FROM
      pt_trmast, pr_trancode, pr_currency, pt_trlink, pr_tradestage, pr_transdat,
      pr_account_view, pr_secname
   WHERE
      (pt_trmast.acid * = pt_trlink.acid1)
      and (pt_trmast.tmid * = pt_trlink.tmid1)
      and (pt_trmast.tmcode * = pr_transdat.tmcode)
      and (pt_trmast.tmcode = pr_trancode.trancode)
      and (pt_trmast.tm_price_ccy_id = pr_currency.smsecid)
      and (pt_trmast.tmcode = pr_tradestage.trd_tmcode)
      and (pt_trmast.acid = pr_account_view.acid)
      and (pt_trmast.tmsecid = pr_secname.smsecid)
      and (pt_trmast.acid in (32121))
      AND (pt_trmast.tm_setlbac_secid1 = 3214)
      AND (pt_trlink.tlreason = 'e'
      OR pt_trlink.tlreason = 'E')
      AND (pr_secname.smdate <= '24 - 7 - 2003 12:19:17.000'
      AND pr_secname.smenddate > '24 - 7 - 2003 12:19:17.000')
      AND (pr_tradestage.trd_trdstg in (1, 2))
      AND (pt_trmast.tmeffdate >= '24 - 6 - 2003 0:0:0.000'
      AND pt_trmast.tmeffdate <= '24 - 7 - 2003 12:19:17.000')
      AND ((1 = 0)
      OR (1 = 1
      AND pt_trmast.cancel_date = '1 - 1 - 1900 0:0:0.000'))
      AND NOT EXISTS (
   select
      *
   from
      pt_trmast tr2
   where
      tr2.tm_expirydate < > '1 - 1 - 1900 0:0:0.000'
      and tr2.tmtradedate <= '24 - 7 - 2003 12:19:17.000'
      and tr2.txtmid = pt_trmast.txtmid)
   ORDER BY pt_trmast.tmeffdate DESC
```

Transactions

Knose can also perform transaction analysis. **A transaction, as defined by Knose, is a set of SQL statements comprising a single logical query unit. This is typical of an action performed by a client process. For example, a Windows client wanting to update some data may need to perform 3 selects, merge the results and then update 2 tables. When the user clicks the OK button on the windows PC, the program connects to the database (unless already connected) and performs 5 SQL statements needed to complete the update. Knose groups these 5 separate statements together analysing them as one transaction, splitting the processing time between the SQL database time, and the PC time. The SQL time is the actual time spent processing the SQL statements in the Sybase database. The PC time is the time spent either in sending the results back to the PC (i.e. network delay) or PC processing time.**

Typically, a client application will wrap the SQL statements for a transaction with BEGIN and COMMIT statements. Mostly, this allows Knose to easily detect these transaction units, however, not all applications use BEGIN..COMMITS, and in these instances, Knose uses a heuristic involving the time between transactions sent by each client.

Continuing with our first example, and using the timing method, we run the command again without the sql switch, allowing whole transactions to be detected, providing the following results:

```
sqlanalyze -v -t sql_200308*.txt > alltrans.txt
```

TOT-SQL	COUNT	SQL	PC	nSQL	CRC	DETAIL
433.207	8	54.151	0.436	20	014e455c112	BEGIN TRANSACTION|EXECUTE gpssp_phydate
309.055	13	23.773	0.339	7	007eb986848	USE hilive|SELECT orr_prt_pty_grp.ppg_s
253.160	675	0.375	0.220	13	00dda53f3ae	BEGIN TRANSACTION|SELECT btc_status FRO
203.042	2	101.521	0.378	7	0075c6f8833	USE hilive|orr_prt_pty_grp.ppg_sty_id,r
199.213	15	13.281	0.531	12	00cf4a88246	EXEC sp_server_info|USE hilive|SELECT o
158.689	1	158.689	0.566	8	008288ce1af	EXEC sp_server_info|USE hilive|orr_prt_
134.031	198	0.677	1.070	9	009629ca2da	EXECUTE gpssp_readadjustdt|SELECT pr_he
125.566	18	6.976	12.646	86	0567dea8caa	SELECT ms_function.mfn_function_name FR
114.782	125	0.918	0.327	16	0109a2837b5	SELECT pr_account_view.acacname,fot_tra
98.953	160	0.618	1.545	5	005b573482e	SELECT oir_batch.btc_batch_id,oir_batch
98.656	226	0.437	0.650	7	007cafc5aae	EXECUTE sdsp_get_systemdat|SELECT pr_se
95.755	123	0.778	1.808	6	006dbc18ac8	SELECT oir_batch.btc_batch_id,oir_batch
94.608	61	1.551	2.958	12	00c09a48c0f	SELECT oir_batch.btc_batch_id,oir_batch
93.788	69	1.359	2.641	11	00b981129ca	SELECT oir_batch.btc_batch_id,oir_batch
93.420	27	3.460	5.857	27	01b3008329e	SELECT oir_batch.btc_batch_id,oir_batch
92.289	227	0.407	0.099	8	00822614c35	EXECUTE gpssp_readadjustdt|SELECT DISTI
91.531	106	0.864	1.929	7	007ff33704d	SELECT oir_batch.btc_batch_id,oir_batch

To analyse one of these transactions further, we can use the –sql switch again to extract the SQL statements in a more readable format:

```
sqlanalyze -v -sql -crc 0109a2837b5 sql_200308*.txt > crc_0109a2837b5.unx
```

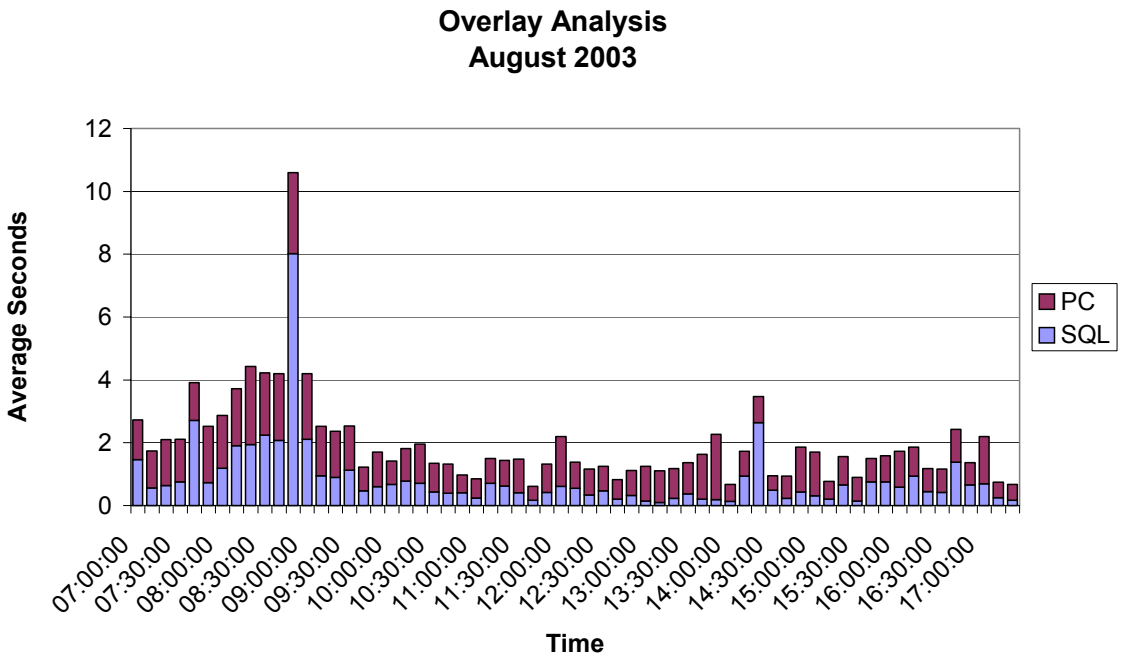
This returns all the SQL for the transaction, with timestamps showing the time taken by each individual query within the transaction unit.

```
2003/08/01 09:52:48.377   0.221 seconds
     SELECT
        pr_account_view.acacname, fot_tranhist.tmcode, fot_tranhist.effdate,
        pr_secname.smshortdesc, pr_secname.smdesc1, fot_tranhist.tmquantity,
        fot_tranhist.tmprice, fot_tranhist.tmbnet, fot_tranhist.tmtotcombas
     FROM
        fot_tranhist, pr_account_view, pr_secname, pr_secmast, pr_acbaseccy,
        pr_secbond, it_violation, fot_proporderbatch
     WHERE
        (pr_secbond.smsecid = * fot_tranhist.secid)
        AND (fot_tranhist.fth_transgroup_id = 828194)
        AND (fot_tranhist.fth_status = 'V'
        OR fot_tranhist.fth_status = 'E')
     ORDER BY pr_account_view.acacname ASC

2003/08/01 09:52:48.717   0.168 seconds
     SELECT pr_currency.smsecid, pr_currency.currcode, pr_currency.currname
     FROM   pr_currency
     WHERE (pr_currency.smsecid < > 0)

2003/08/01 09:52:49.246   0.004 seconds
     SELECT
        pr_tranfunc2.ordertype, pr_tranfunc2.trancode, pr_tranfunc2.smtype,
        pr_tranfunc2.clientfunc
     FROM
        pr_tranfunc2
     WHERE
        (pr_tranfunc2.clientfunc = "ACME")
        and (pr_tranfunc2.smtype = "500")
        or (pr_tranfunc2.clientfunc = "FLOO"
        and (pr_tranfunc2.smtype = "000"))

2003/08/01 09:52:56.285   7.037 seconds
     SELECT
        pr_account_view.acacname, fot_tranhist.tmcode, fot_tranhist.effdate,
```

```
      pr_secmast.smtype, fot_tranhist.fth_aiunit, pr_secmast.sm_lotsize,
      externalcode1, " " as externalcode2, " " as externalcode3, "" as c_longshort
      pr_secbond.matdate, fot_tranhist.fth_transmit_ref, 0 as bulkno,
      pr_account_view.acclifund, pr_account_view.acm_com_par_id,
      fot_proporderbatch.pob_mgi_id
   FROM
      fot_tranhist, pr_account_view, pr_secname, pr_secmast, pr_acbaseccy,
      pr_secbond, it_violation, fot_proporderbatch
   WHERE
      (pr_secbond.smsecid = * fot_tranhist.secid)
      and (it_violation.igv_txtmid = * fot_tranhist.fth_transmit_ref)
      and (it_violation.igv_acid = * fot_tranhist.acid)
      and (fot_tranhist.effdate < pr_secname.smenddate)
      AND (fot_tranhist.fth_transgroup_id = 123123)
      AND (fot_tranhist.fth_status = 'V'
      OR fot_tranhist.fth_status = 'E')
   ORDER BY pr_account_view.acacname ASC

TOTAL 7.947 seconds
```

Overlay Analysis

Knose can produce an overlay report, particularly useful for finding hotspots in a day's activities. For example, a large batch job running at certain times during the day could be causing generally poor system performance. To assist in detecting (and isolating) such a case, the "–overlay" switch overlays several days' data onto a 24 hour period. The following graph shows the overlay analysis for a HiInvest application running over a month. Hotspots can be seen during the morning from 08h00 to 09h00, and a possible peak at 14h30.

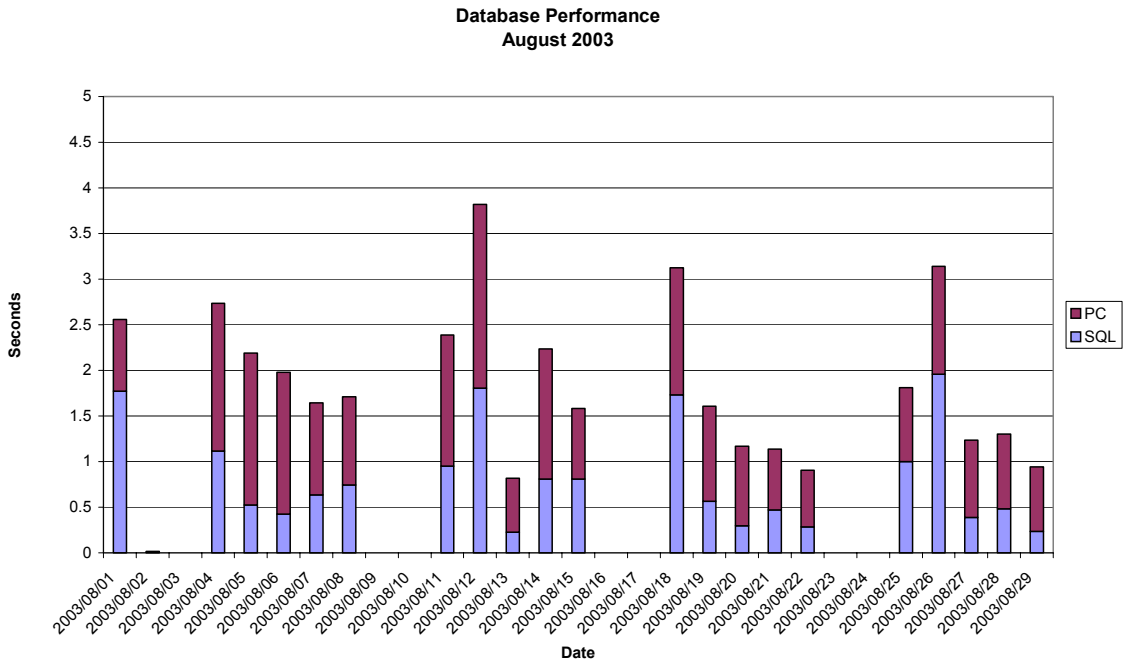


User Analysis

User analysis provides a breakdown of SQL usage by IP address (mapping to user names if a host file is provided). The format of the user report is as follows.

DATE	IP-ADDRESS	USERNAME	TOTAL	AVERAGE	SQL	PC	SQL-SLOW	PC-SLOW
20030820	10.190.14.26	Peter_S	75	2.36636	1.40401	0.96235	15	19
20030820	10.190.13.70	Anne_Michaels	346	1.57119	0.36733	1.20387	28	135
20030820	10.190.20.36		151	1.76751	0.11977	1.64774	1	49
20030820	10.190.19.43	Hendrick_T	316	0.73978	0.16753	0.57224	5	75
20030820	10.190.20.35		131	2.00067	0.14427	1.85640	3	56
20030820	10.190.19.48	PC2246	224	0.80628	0.13134	0.67493	0	54
20030820	10.190.2.21		22	6.16889	5.66160	0.50729	12	5
20030820	10.190.20.52	Jamie_Smythe	108	0.80984	0.30059	0.50925	4	22
20030820	10.190.16.102		60	1.01599	0.14847	0.86752	0	20
20030820	10.190.20.63	Japie_van_Rooyen	3	20.25624	20.21596	0.04029	1	0
20030820	10.190.19.35	PC3298	249	0.51129	0.11237	0.39892	2	36
20030820	10.190.18.54		49	0.62917	0.06572	0.56345	0	14
20030820	TOTAL		2220	1.16891	0.29653	0.87238	75	576

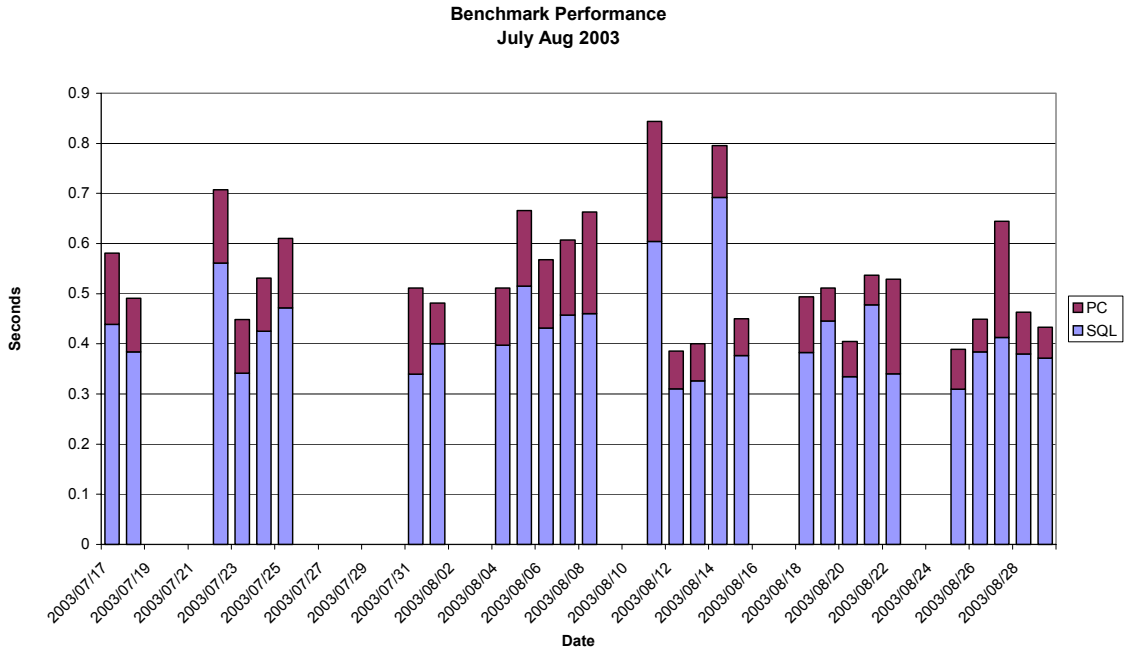
This report is useful for analysing database usage on a per client basis. If a user analysis is performed on a daily basis, then it is fairly simple to extract data for a particular user, or for all the users, and provide overall performance graphs for a week or a month.



Benchmark Analysis

The previous analysis provides a clear indication of overall system performance, but can be skewed by large ad hoc jobs. To provide a more consistent measure of performance, the previous analysis can be performed using only one transaction type. By specifying the transaction id to the sqlanalyze tool, the figures are then based on the same transaction, which under ideal conditions, would always complete in the same amount of time.

The graphs produced using the benchmarks give an accurate indication of the actual system performance. Clearly, the choice of benchmark transaction is important, and a mix of transactions should be chosen to measure various aspects of the system.



Solving performance problems, quickly

Performance tuning often remains more of an art than a science and without the program source at hand hunting down increasingly inefficient SQL processes within a multi-user, live environment is an almost impossible task. The Knose Tools provide essential data for quickly isolating and fixing problem queries run by various database applications.