Sponsors of International i-Power 2019

International i-Power 2019

June 11th to June 12th

Milton Keynes

16 Ways of Accessing Data by Key

Kaare Plesner

© iPerformance ApS, Denmark

My Background in Brief

Began working with this platform in the System/38 era

Dedicated performance aficionado

Taught courses in AS/400 performance for IBM
+10 years +20 countries 5 languages

15 years “No cure – no pay” optimization

Director of iPerformance ApS, developer and supplier of GiAPA® → software using AI to analyze application performance

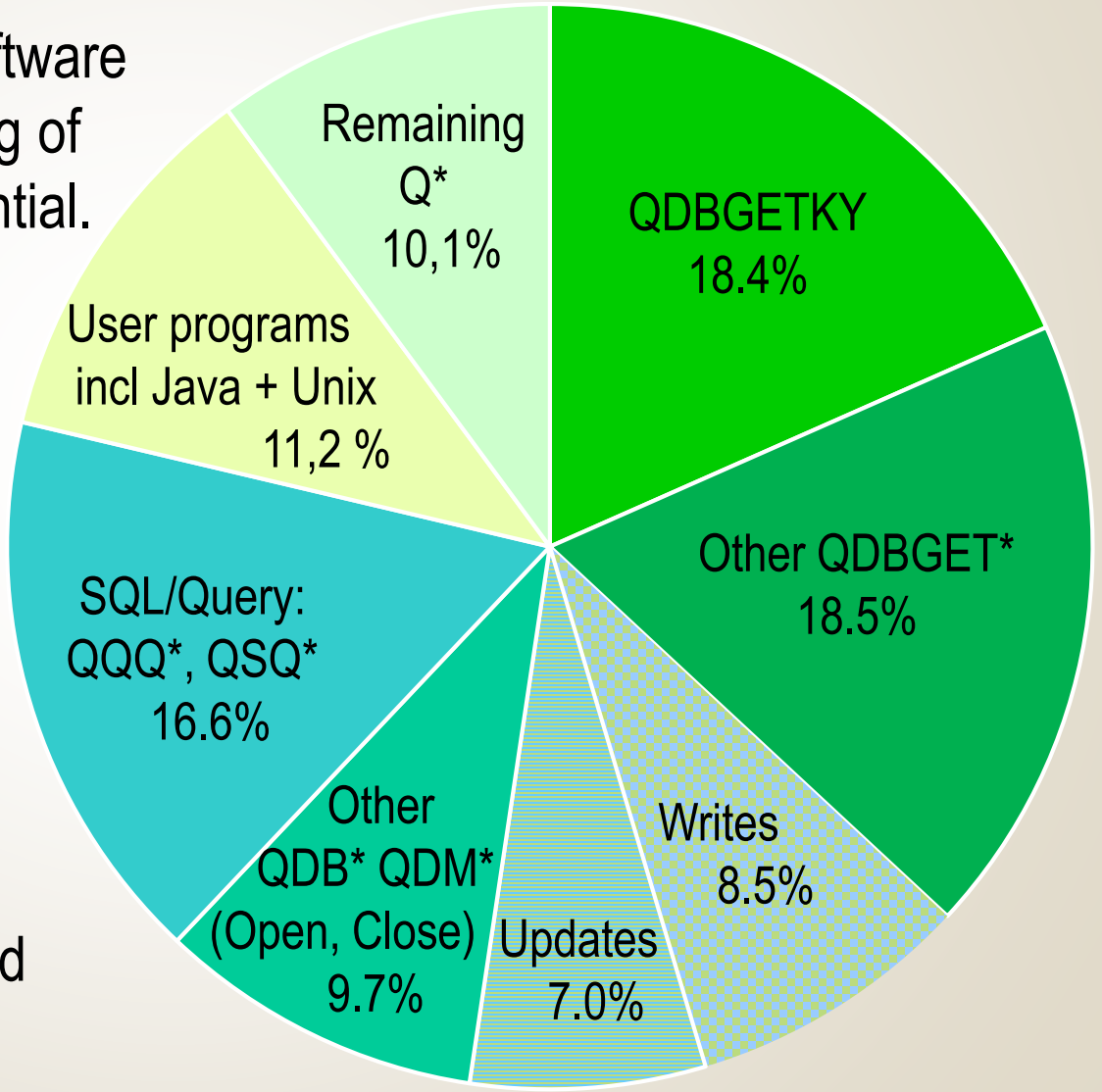
Why Analyze Read-by-Key?

We wanted to add **AI** to our software → enable automatic pinpointing of performance optimization potential.

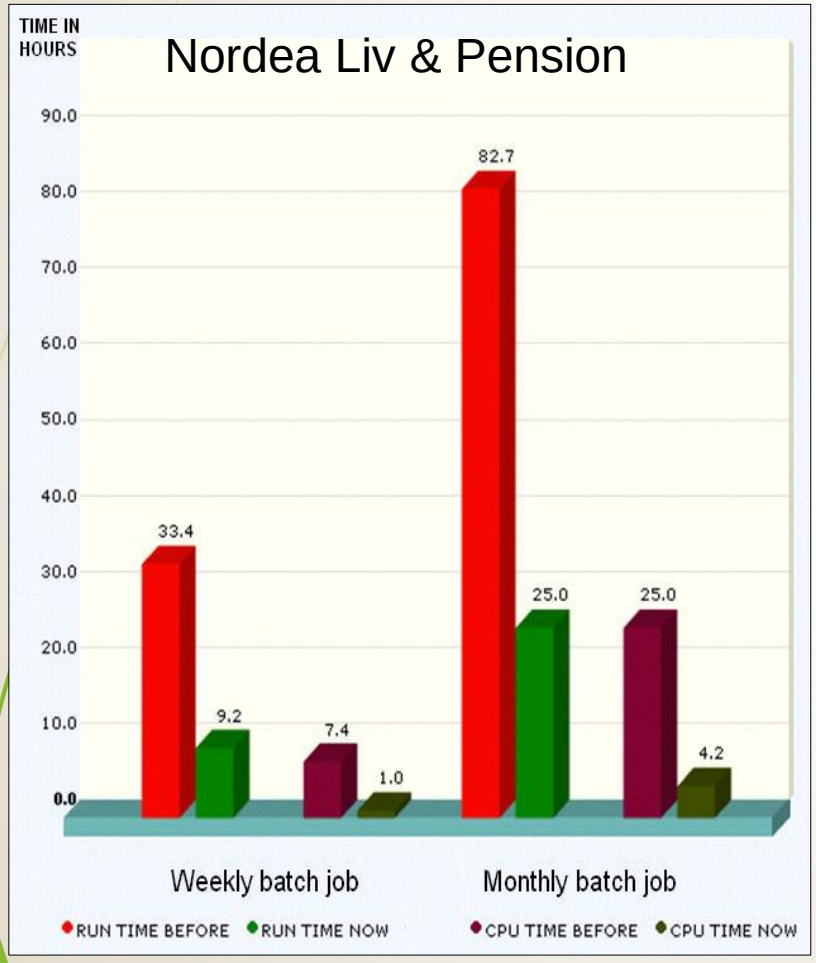
3.8 million call stacks retrieved automatically by GiAPA from > 50 servers worldwide were analyzed:

Reading data accounted for > 40% of resources used.

Access methods used appeared very often to be less efficient.



Is it Worth the Effort?



Largest European Power i Site:

New batch application job took **4½ hours** for one division.

Issue: They had 30+ divisions.

Solution: Reduce reads of data by changing access methods.

Result:
43 min run time per division

Preparation for Test

Files/Tables defined:

1. PF CUSTOM_DDS, + LF having customer number as key, containing fields like customer address, sales district, amounts, etc.
2. SQL-Table CUSTOM_SQL, + SQL index with customer number as key.
3. SQL-Table TRANS_SQL (traditional sales transactions).

Records added using "native" I/O:

262144 records in DDS-defined CUSTOM_DDS.
Customer number initiated with values from 0 to 262143.

Rows added using SQL:

1. CUSTOM_DDS copied into SQL-defined table CUSTOM_SQL.
2. TRANS_SQL having 5 million rows with column "Customer Number" containing a random value between 0 and 262143.

“Rules of the Game”

All programs use the same algorithm to calculate the “random” key used for 5.000.000 accesses to the file or table – only access method differs.

Each program accesses all test records/rows 19 times.

Resource usage collected for each test:

- Elapsed time based on time stamps
- CPU seconds
- Physical DB I/Os

“Total sales” reported per test verifies that the same records were accessed

Test Environment

Power i model 720 8202 E4D, one processor, 16 GB memory, V7R3M0.

Jobs active:

1. Our test job running 16 programs, each using different access methods to fetch data from the “Customer Master” 5 million times.
2. GiAPA software (Global i Application Performance Analyzer) collecting the test job call stack twice per second (CPU usage = 0.08%).

None of the read test runs showed any disk I/Os for the files/tables.

CPU usage averaged 54% during all tests (100% is not possible).

1

Input file / table: **CUSTOM_DDS**

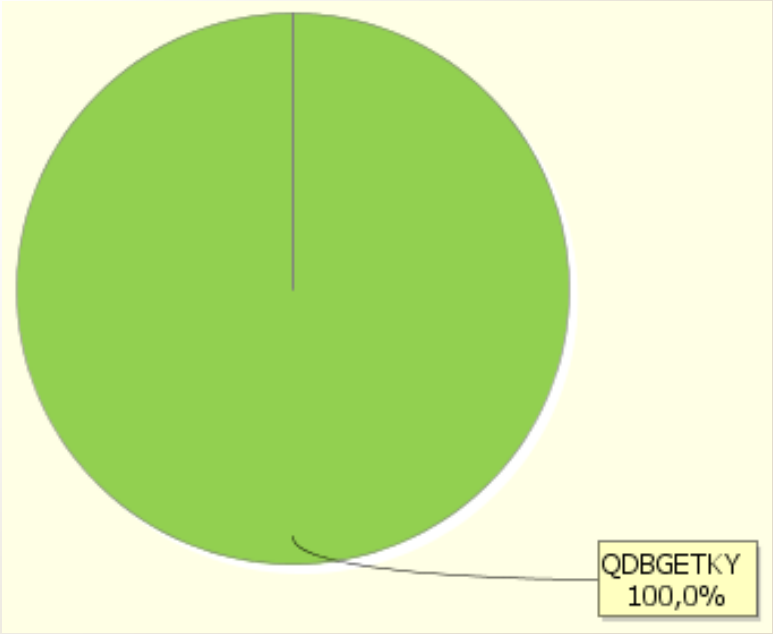
Access method: **Regular "native" I/O:**

RPG **CHAIN**

COBOL **READ (Organization indexed, Record Key is ….)**

	Seconds used	Percent of #1
Elapsed time	34.1	100
CPU	19.1	100

Call stack statistics on active program is fetched twice per second:

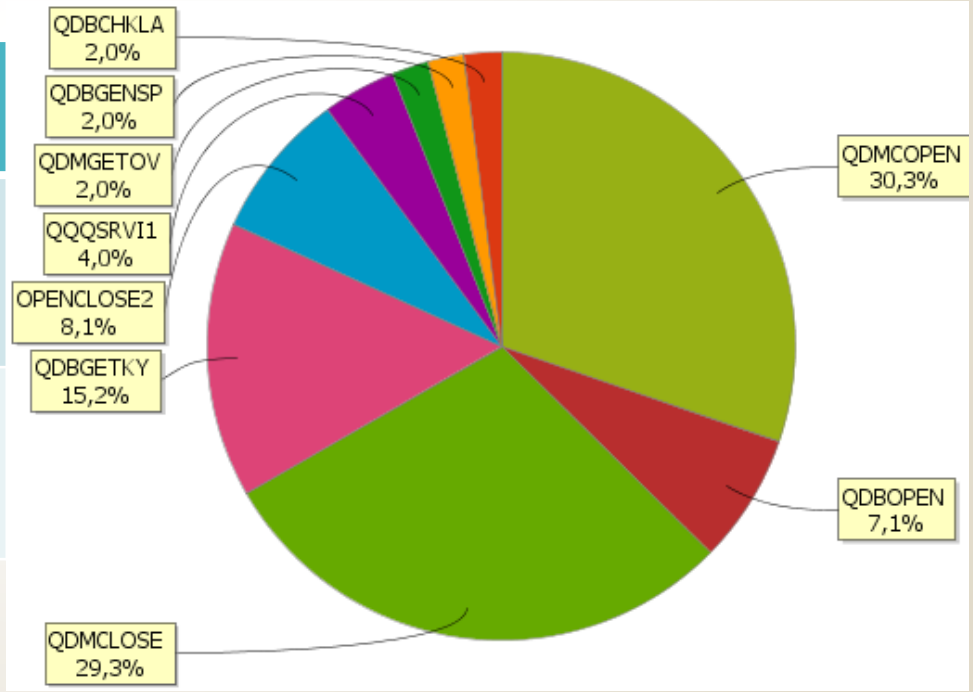


2

Input file / table: **CUSTOM_DDS**

Access method: **Call to subprogram using native read by key.**
Subprogram is closed after each call.
(RPG LR *On, COBOL STOP RUN)

	Seconds used	Percent of #1
Elapsed time	765.4	2,248
CPU	427.4	2,236

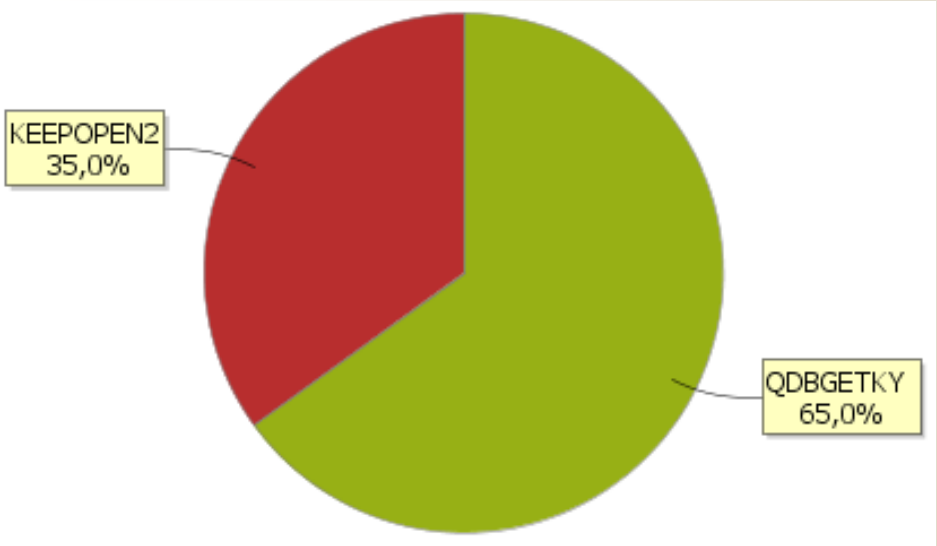


3

Input file / table: **CUSTOM_DDS**

Access method: **Call to subprogram using native read by key. Subprogram and file is kept open.**
(RPG not LR *On, COBOL EXIT PROGRAM or GOBACK)

	Seconds used	Percent of #1
Elapsed time	49.1	144.1
CPU	27.5	143.8

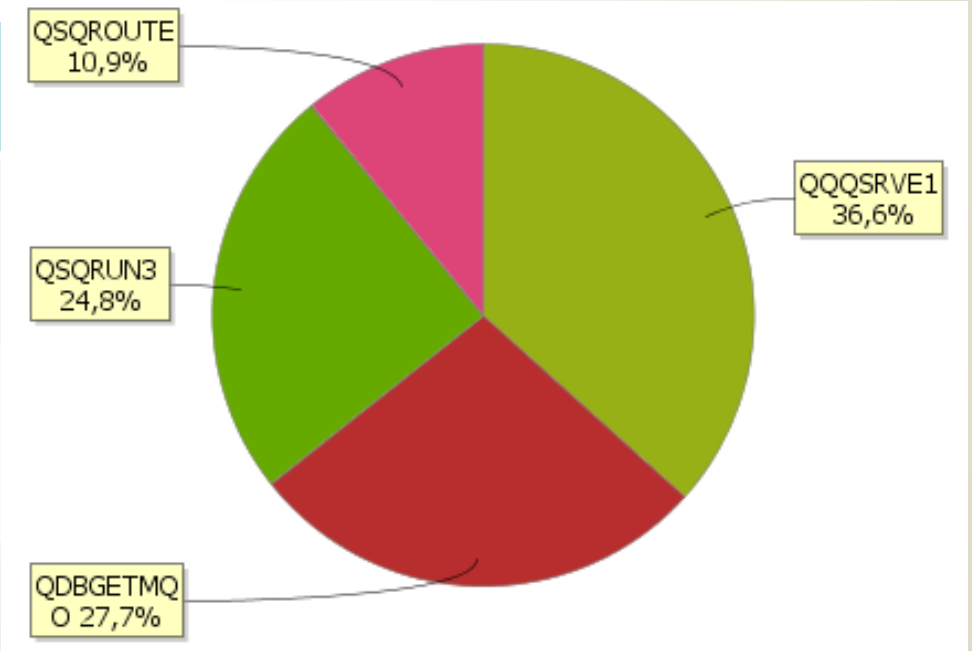


4

Input file / table: CUSTOM_DDS

Access method: Embedded SQL: Select * into :InputRec
from CUSTOM_DDS
where CUSTNO = :KeyForRead

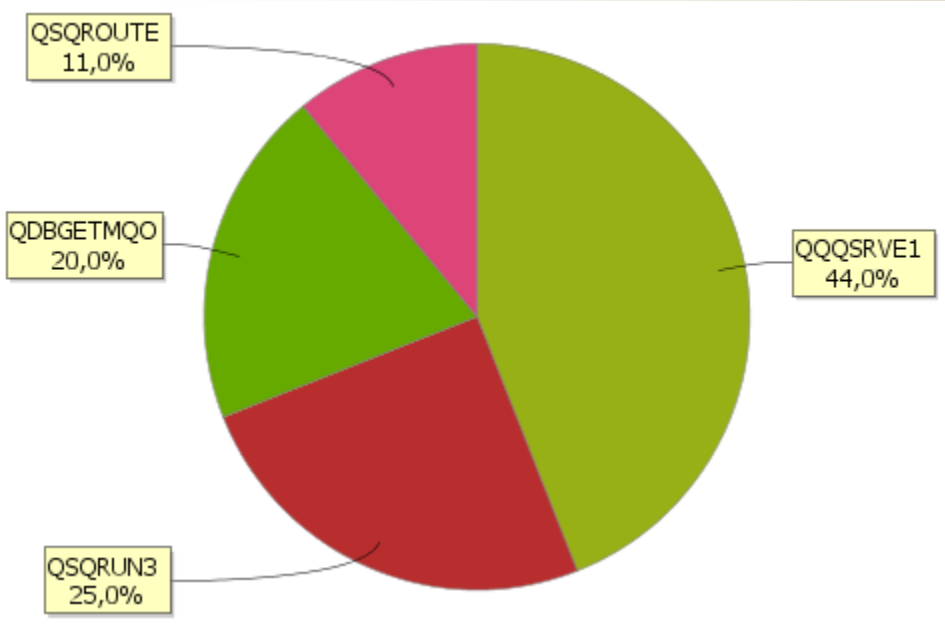
	Seconds used	Percent of #1
Elapsed time	148.7	436.7
CPU	82.3	430.4



Input file / table: **CUSTOM_SQL**

Access method: **Embedded SQL: Select * into :InputRec**
from CUSTOM_SQL
where CUSTNO = :KeyForRead

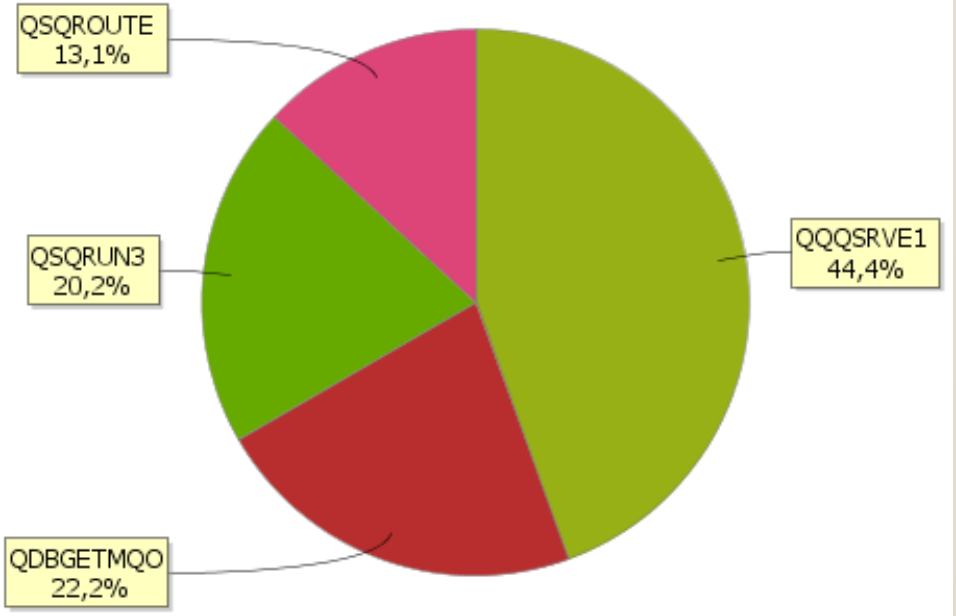
	Seconds used	Percent of #1
Elapsed time	147.5	433.3
CPU	81.6	426.9



Input file / table: **CUSTOM_DDS**

Access method: **Embedded SQL: Like #4, but Select limited to the four columns required by the program**

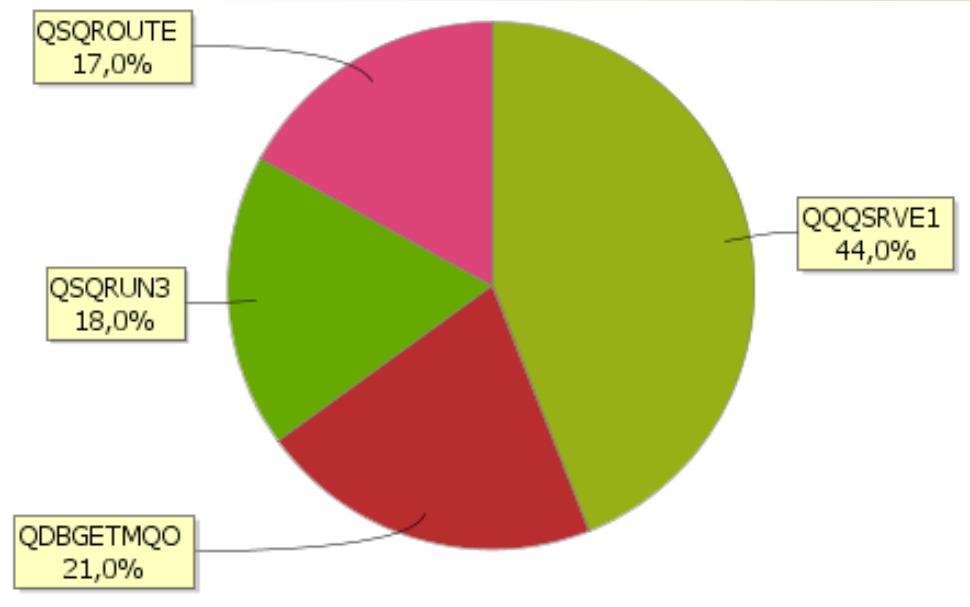
	Seconds used	Percent of #1
Elapsed time	153.8	451.6
CPU	85.0	444.8



Input file / table: **CUSTOM_SQL**

Access method: **Embedded SQL: Like #5, but Select limited to the four columns required by the program**

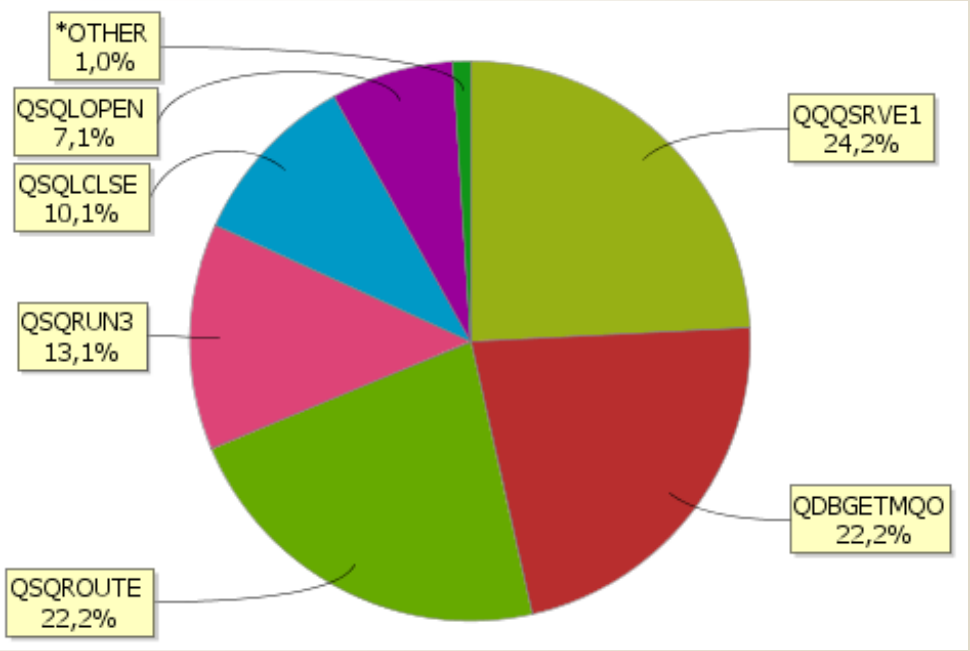
	Seconds used	Percent of #1
Elapsed time	154.8	454.5
CPU	85.6	447.7



Input file / table: **CUSTOM_DDS**

Access method: **Embedded SQL: Prepare and Select only the four columns required**

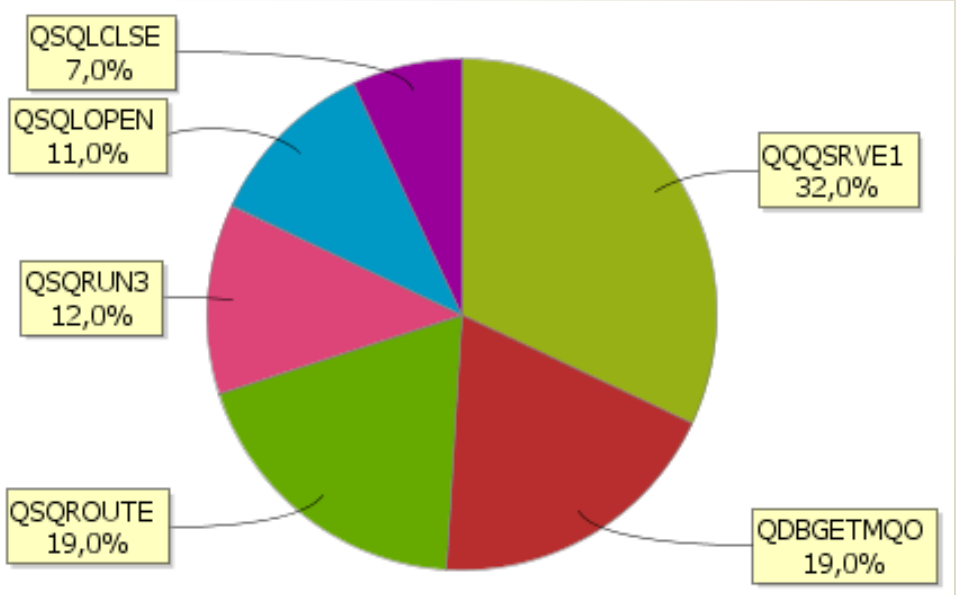
	Seconds used	Percent of #1
Elapsed time	212.0	622.4
CPU	117.2	613.3



Input file / table: **CUSTOM_SQL**

Access method: **Embedded SQL: Prepare and Select only the four columns required**

	Seconds used	Percent of #1
Elapsed time	213.1	625.8
CPU	117.9	616.6



Preparation for #10

Create index TSTREADPFR/CUSTOMERL2
on TSTREADPFR/CUSTOM_SQL
(CustNo, SalesThisY, NbrOfOrders, CustName, CustZip, CustCat)

	Seconds used	Percent of #1
Elapsed time	0.6	1.8
CPU	0.2	1.1

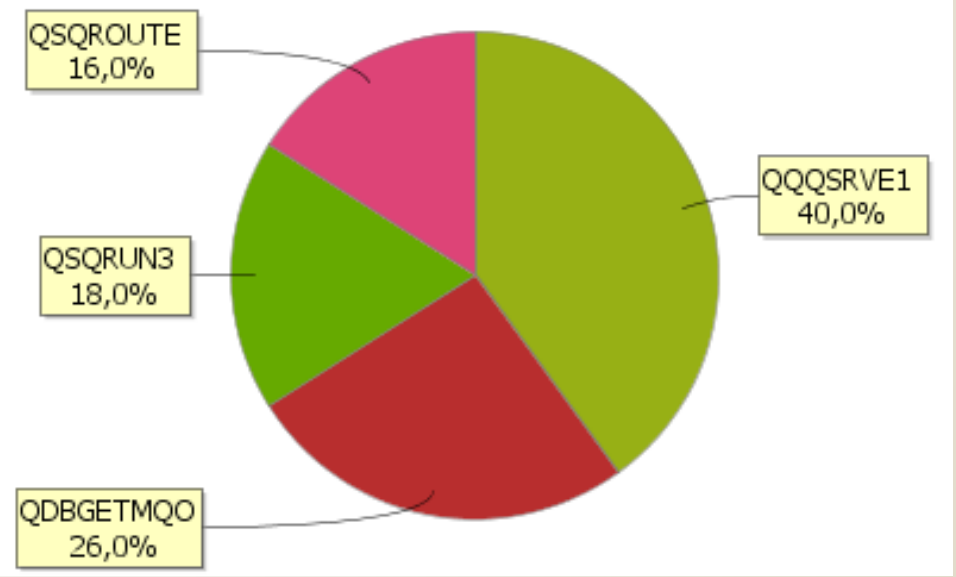
"Call Stack Not Available" during index generation

We enable SQL to use "Index-only-access" → No need to access records

Input file / table: **CUSTOM_SQL**

Access method: **Embedded SQL: Select only four columns using index-only-access**

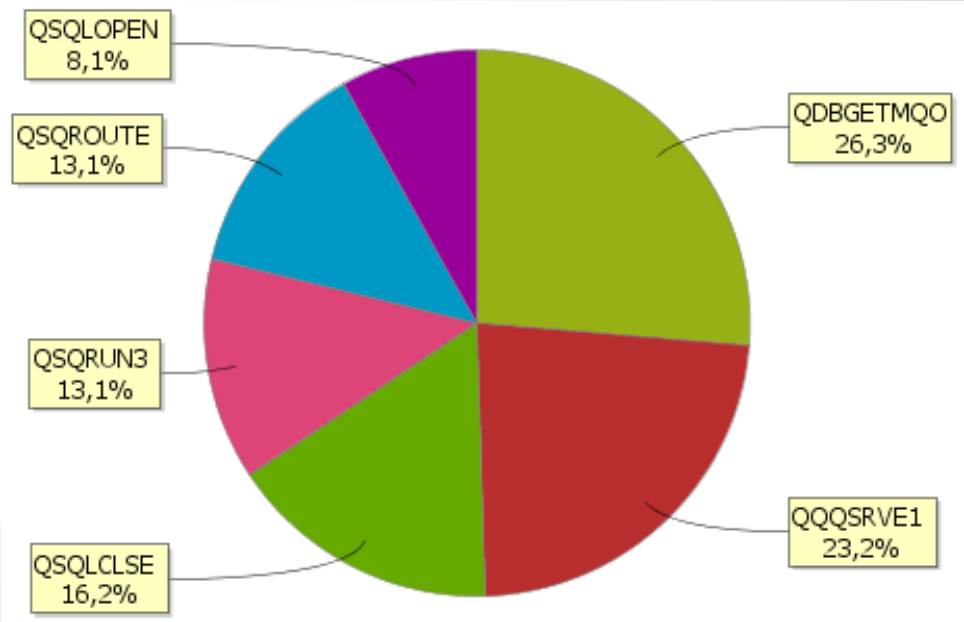
	Seconds used	Percent of #1
Elapsed time	153.2	449.8
CPU	84.4	441.5



Input file / table: **CUSTOM_SQL**

Access method: **Embedded SQL: Prepare & select four columns using index-only-access**

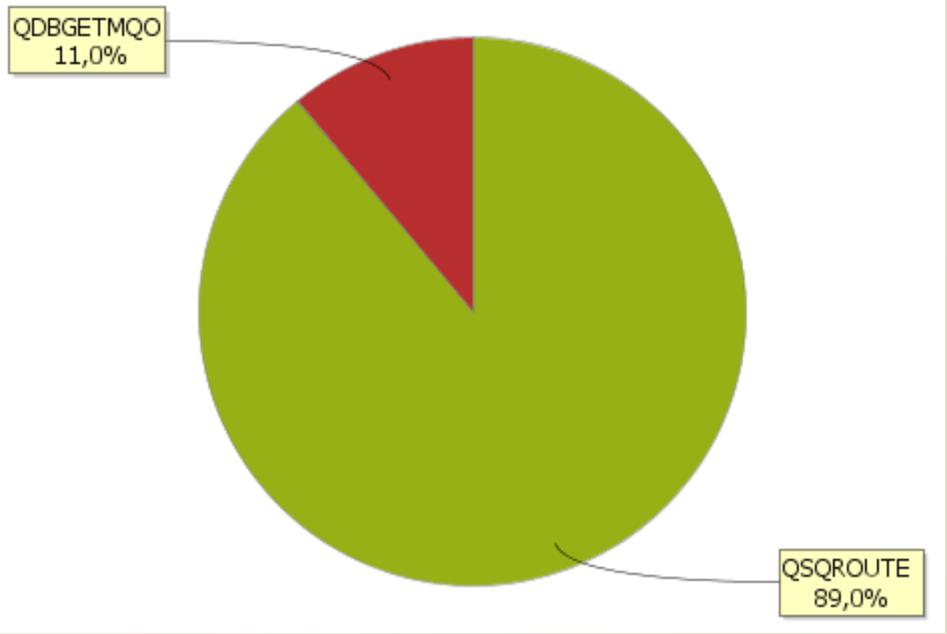
	Seconds used	Percent of #1
Elapsed time	203.1	596.4
CPU	112.3	587.6



Input files / tables: CUSTOM_SQL and TRANS_SQL

Access method: Embedded SQL: Key for reading CUSTOM_SQL origins from sequentially read table TRANS_SQL

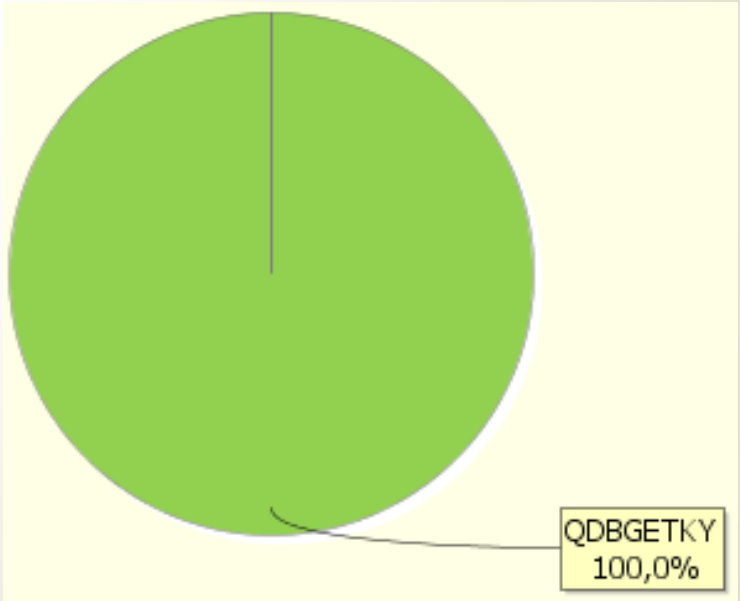
	Seconds used	Percent of #1
Elapsed time	41.4	121.5
CPU	22.9	119.7



Input files / tables: **CUSTOM_SQL and TRANS_SQL**

Access method: **Traditional "native" I/O:**
(RPG CHAIN, COBOL READ with Org. indexed)
Key for reading CUSTOM_SQL origins from sequentially read file TRANS_SQL

	Seconds used	Percent of #1
Elapsed time	34.8	102.1
CPU	19.5	102.2

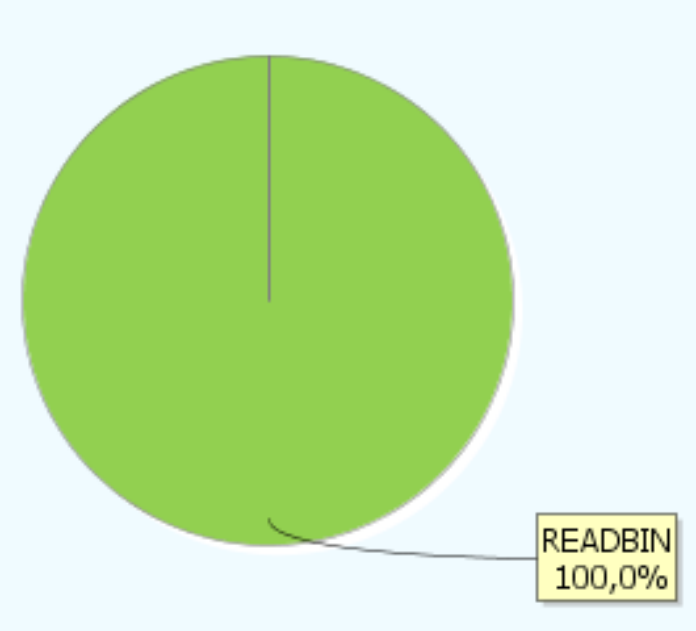


Input file / table: **CUSTOM_DDS → User Space array**
(Loading took 0.4 seconds, 78 ms CPU)

Access method: **Binary table lookup.** COBOL: SEARCH ALL
RPG: %LOOKUP in array defined with ASCEND

	Seconds used	Percent of #1
Elapsed time	7.8	21,4
CPU	4.3	22.8

Time shown includes load of table



(Sequential lookup used 5 hours 19 min. elapsed, 3 hours 2 min. CPU!)

Preparation for #15

Read entire CUSTOM_DDS file, load the key and four data fields used into a User Index → "Our own index-only-access"

	Seconds used	Percent of #1
Elapsed time	0.9	2.9
CPU	0.5	2.6

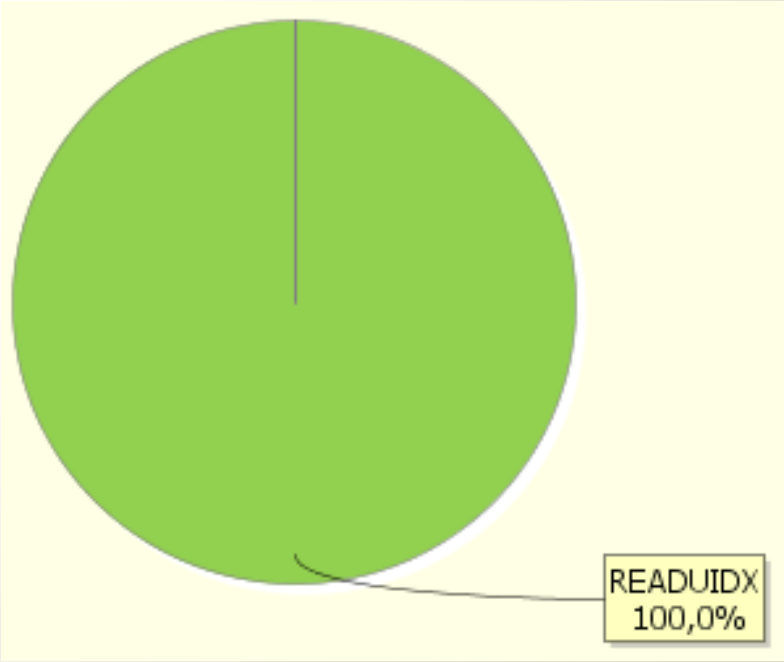
NOTE: "Difficult" or "Advanced" technically means "Something I haven't learned yet …"

Input file / table: **None (we use pre-loaded User Index)**
(Alternative: Load used records on the fly)

Access method: **MI-instruction FNDINXEN (Find Index Entry), available via C Function Library**

	Seconds used	Percent of #1
Elapsed time	8.4	24,6
CPU	4.7	24.7

20 times faster than SQL Index-only-access



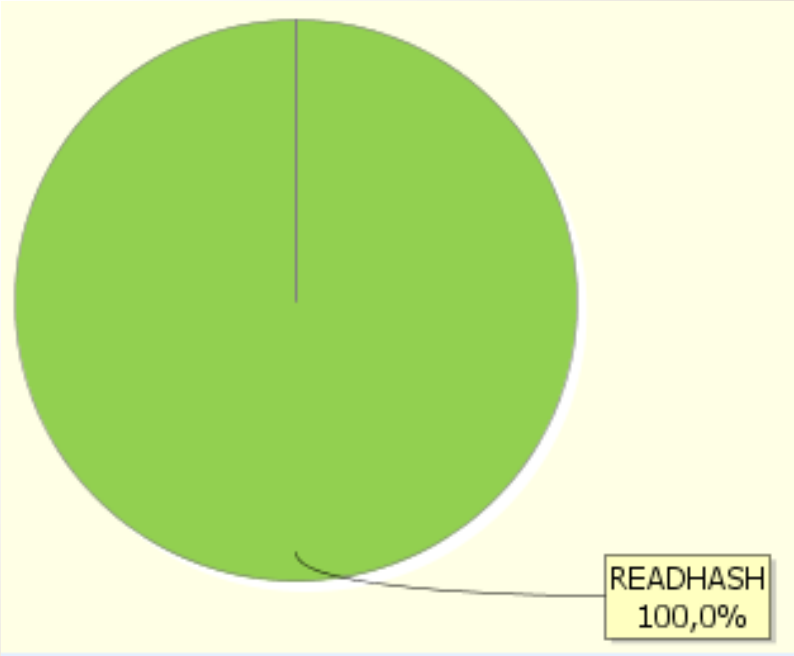
Input file / table: **CUSTOM_DDS read sequentially blocked and loaded into memory-table**

Access method: **“Look-up” in hash table in a (user) space**

(Hash-Table → Pointer to wanted data is calculated based on key value)

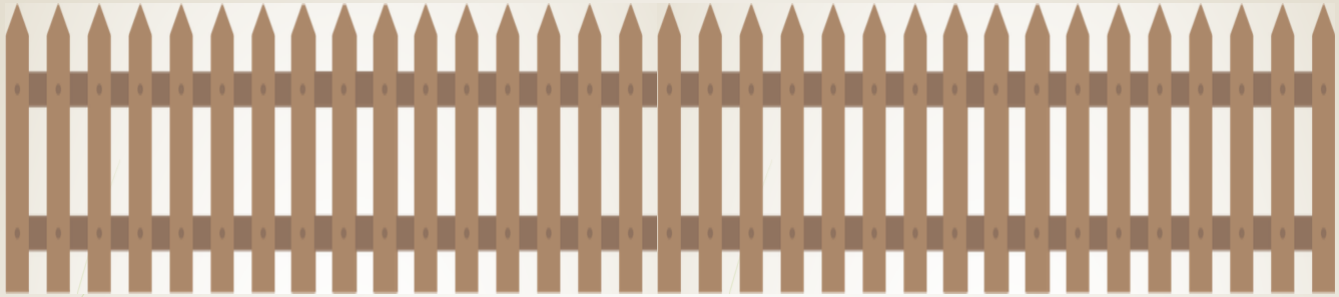
Including Table load	Seconds used	Percent of #1
Elapsed time	2.0	5.9
CPU	1.0	5.2

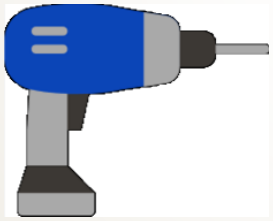
Time shown includes load of table



Overview of Results of Read-by-key Test Runs

Elapsed mm:ss.s	Relativ pct.	CPU msec.	Relativ pct.	File used	Description of Action completed by job	Control total: "Sales this year"
34.1	100	19.116	100	DDS	Keyed reads completed	1,722,908,677,250
12:45.4	2,247	427.422	2,236	DDS	Subpgm reads + sets LR on	1,722,908,677,250
49.1	144	27.496	144	DDS	Subpgm reading kept open	1,722,908,677,250
2:28.7	437	82.269	430	DDS	SQL Fetch of entire record	1,722,908,677,250
2:27.5	433	81.610	427	SQL	SQL Fetch of entire record	1,722,908,677,250
2:33.8	452	85.025	445	DDS	SQL Fetch Selecting only 4 fields	1,722,908,677,250
2:34.8	455	85.589	447	SQL	SQL Fetch Selecting only 4 fields	1,722,908,677,250
3:32.0	622	117.230	613	DDS	SQL Prepare + Select of 4 fields	1,722,908,677,250
3:33.1	626	117.866	617	SQL	SQL Prepare + Select of 4 fields	1,722,908,677,250
0.6		.203		SQL	Created "IndexOnly" suited SQL index	
2:33.2	450	84.398	442	SQL	SQL Fetch selecting only 4 fields	1,722,908,677,250
3:23.1	596	112.322	588	SQL	SQL Prepare + Select 4 fields	1,722,908,677,250
1.0		.531			Transaction file created	
41.4	122	22.876	120	SQL	SQL using cursor with join	1,722,908,677,250
34.8	102	19.538	102	DDS	Chain with key from 2nd file	1,722,908,677,250
7.8	22	4.342	23	DDS	Load + Access binary table in UsrSpc	1,722,908,677.250
0.9		.499			USRIDX created and loaded with data	
8.4	25	4.730	25	None	Access through USRIDX Find completed	1,722,908,677,250
2.0	6	.997	5	None	Load + Access hash table in user space	1,722,908,677,250





Conclusions so far

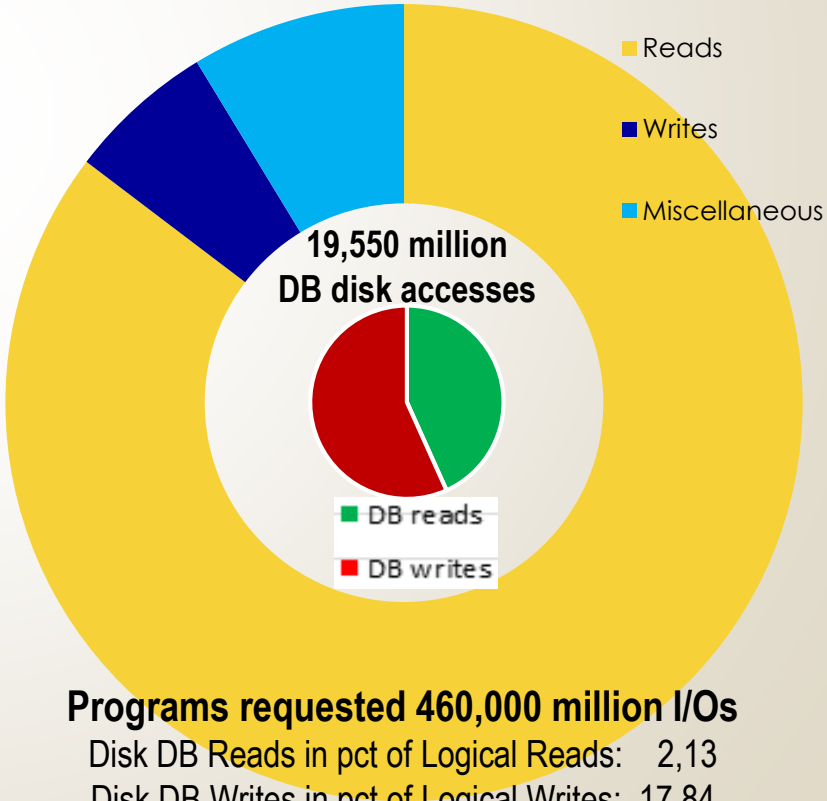
DB I/Os normally always use the lion's share of all resources.

Efficiency of SQL surpasses native I/Os when used for what it was designed to do.

Only consider optimization when I/O counts are approaching millions per day.

(OPTIMIZE(*FULL) had no effect.)

Representative overview of total physical + logical DB I/Os from seven large servers over a ~week.



Programs requested 460,000 million I/Os

Disk DB Reads in pct of Logical Reads: 2,13

Disk DB Writes in pct of Logical Writes: 17,84

"Optimization is a Complex Task" → false

Modifying program file access is neither difficult nor very time consuming:
The main logic of the program remains intact – we only replace the access method.

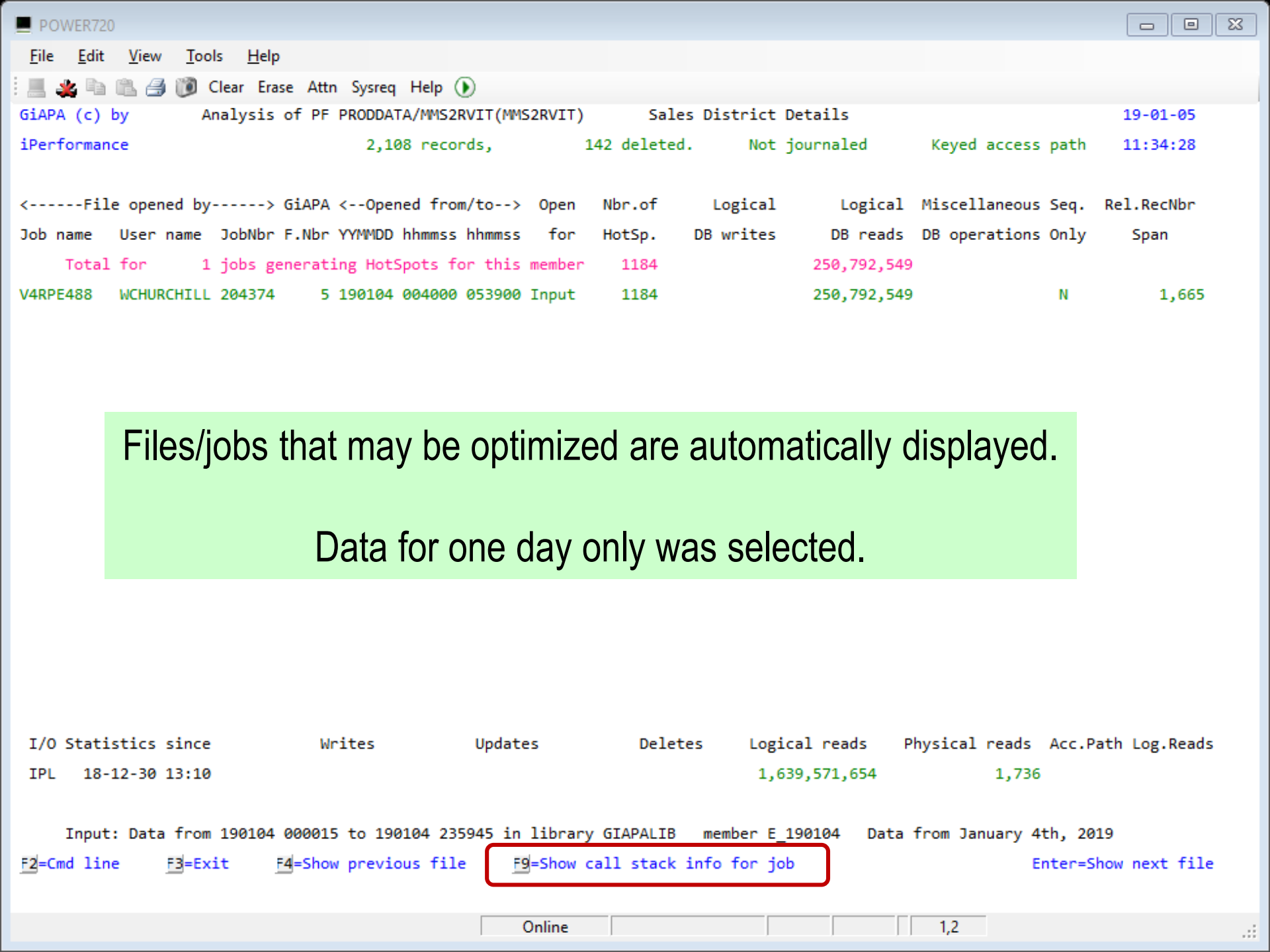
Coding the creation and loading of a User Index is an easy task.
(Complete RPG code example may be requested from kp@giapa.com).

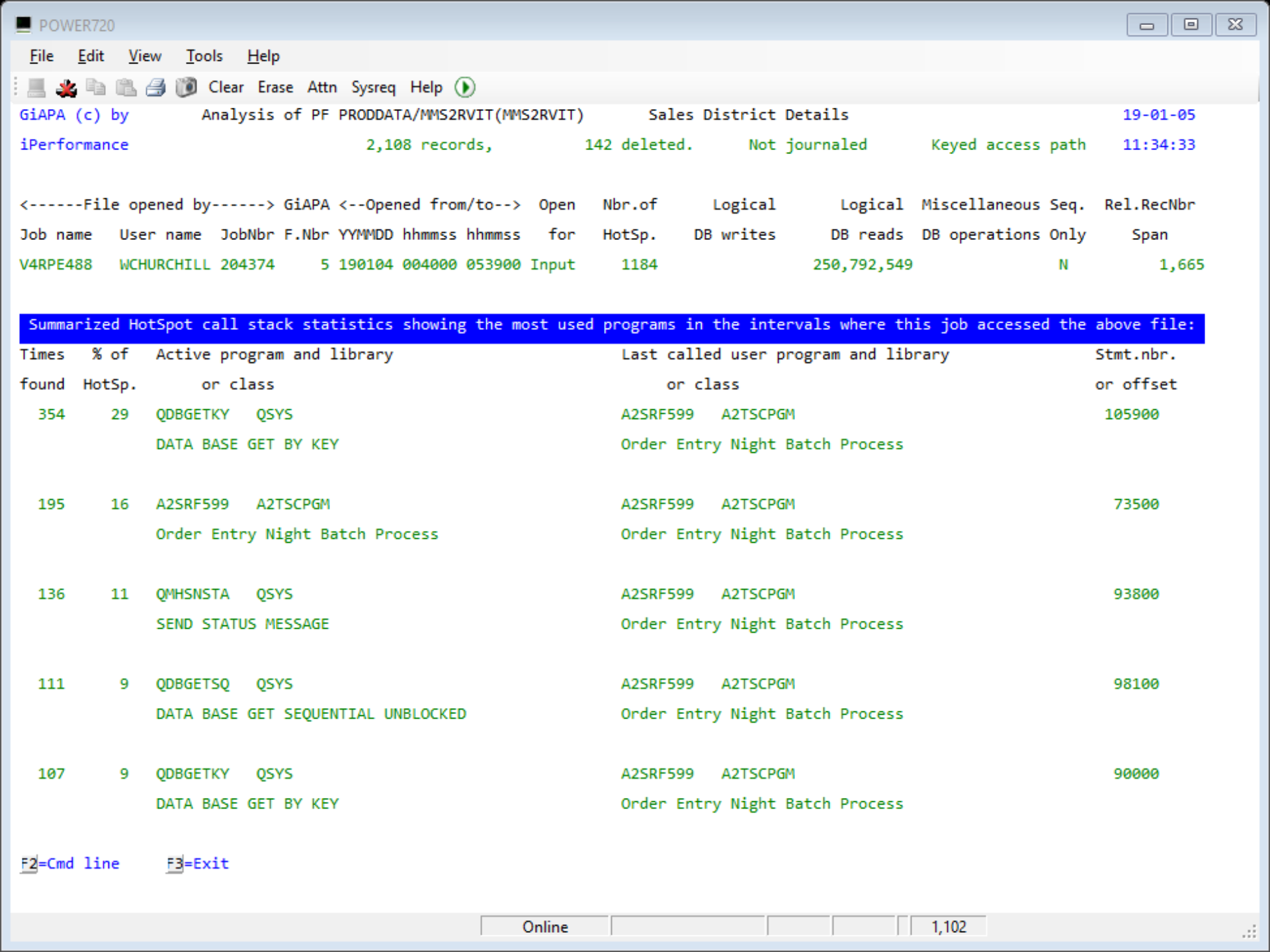
A User Index employs main memory efficiently: Only fields included = necessary columns.
NOTE: **a User Index is a "permanent object"** – it can be saved and restored.

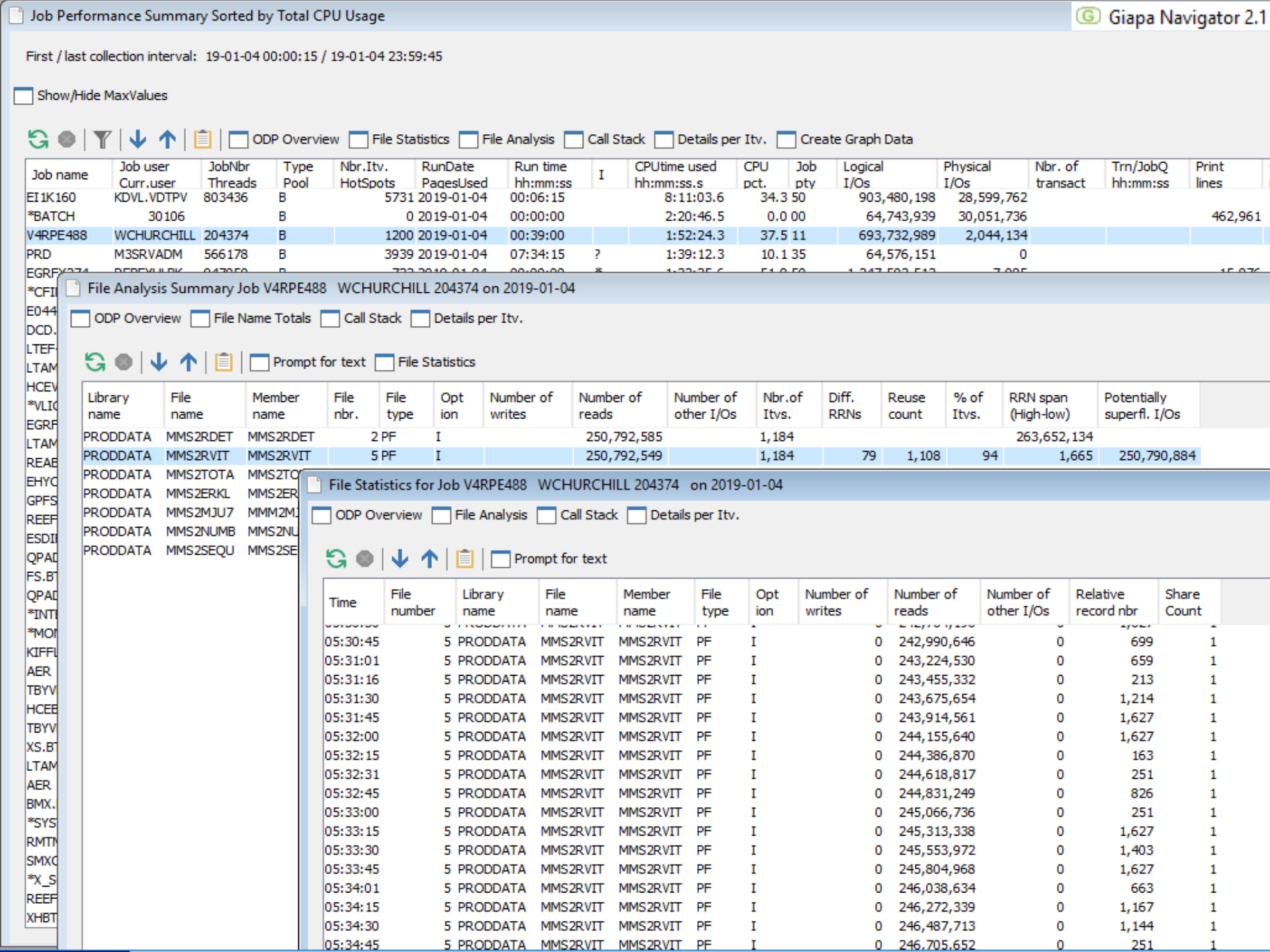
"Locating Optimization Candidates is a Complex Task" → debatable

Next slides exemplify:

1. How our software presents optimization candidates
2. A "Do it yourself" instruction – overall straightforward (and possibly bit time consuming).


POWER720
File Edit View Tools Help
Clear Erase Attn Sysreq Help
GiAPA (c) by Analysis of PF PRODDATA/MMS2RVIT(MMS2RVIT) Sales District Details 19-01-05
iPerformance 2,108 records, 142 deleted. Not journaled Keyed access path 11:34:28
<------File opened by------> GiAPA <--Opened from/to--> Open Nbr.of Logical Logical Miscellaneous Seq. Rel.RecNbr
Job name User name JobNbr F.Nbr YYMMDD hhmmss hhmmss for HotSp. DB writes DB reads DB operations Only Span
Total for 1 jobs generating HotSpots for this member 1184 250,792,549
V4RPE488 WCHURCHILL 204374 5 190104 004000 053900 Input 1184 250,792,549 N 1,665
Files/jobs that may be optimized are automatically displayed.
Data for one day only was selected.
I/O Statistics since Writes Updates Deletes Logical reads Physical reads Acc.Path Log.Reads
IPL 18-12-30 13:10 1,639,571,654 1,736
Input: Data from 190104 000015 to 190104 235945 in library GIAPALIB member E_190104 Data from January 4th, 2019
F2=Cmd line F3=Exit F4=Show previous file F9=Show call stack info for job Enter=Show next file
Online
1,2


POWER720
File Edit View Tools Help
Clear Erase Attn Sysreq Help
GiAPA (c) by Analysis of PF PRODDATA/MMS2RVIT(MMS2RVIT) Sales District Details 19-01-05
iPerformance 2,108 records, 142 deleted. Not journaled Keyed access path 11:34:33
<------File opened by------> GiAPA <--Opened from/to--> Open Nbr.of Logical Logical Miscellaneous Seq. Rel.RecNbr
Job name User name JobNbr F.Nbr YYMMDD hhmmss hhmmss for HotSp. DB writes DB reads DB operations Only Span
V4RPE488 WCHURCHILL 204374 5 190104 004000 053900 Input 1184 250,792,549 N 1,665
Summarized HotSpot call stack statistics showing the most used programs in the intervals where this job accessed the above file:
Times found % of HotSp. Active program and library or class Last called user program and library or class Stmt.nbr. or offset
354 29 QDBGETKY QSYS DATA BASE GET BY KEY A2SRF599 A2TSCPGM Order Entry Night Batch Process 105900
195 16 A2SRF599 A2TSCPGM Order Entry Night Batch Process A2SRF599 A2TSCPGM Order Entry Night Batch Process 73500
136 11 QMHSNSTA QSYS SEND STATUS MESSAGE A2SRF599 A2TSCPGM Order Entry Night Batch Process 93800
111 9 QDBGETSQ QSYS DATA BASE GET SEQUENTIAL UNBLOCKED A2SRF599 A2TSCPGM Order Entry Night Batch Process 98100
107 9 QDBGETKY QSYS DATA BASE GET BY KEY A2SRF599 A2TSCPGM Order Entry Night Batch Process 90000
F2=Cmd line F3=Exit
Online
1,102


Job Performance Summary Sorted by Total CPU Usage
Giapa Navigator 2.1
First / last collection interval: 19-01-04 00:00:15 / 19-01-04 23:59:45
Show/Hide MaxValues
ODP Overview
File Statistics
File Analysis
Call Stack
Details per Itv.
Create Graph Data
Job name
Job user Curr.user
JobNbr Threads
Type Pool
Nbr.Itv. HotSpots
RunDate PagesUsed
Run time hh:mm:ss
I
CPUtime used hh:mm:ss.s
CPU pct.
Job pty
Logical I/Os
Physical I/Os
Nbr. of transact
Trn/JobQ hh:mm:ss
Print lines
EI1K160 KDVL.VDTPV 803436 B 5731 2019-01-04 00:06:15 8:11:03.6 34.3 50 903,480,198 28,599,762
*BATCH 30106 B 0 2019-01-04 00:00:00 2:20:46.5 0.0 00 64,743,939 30,051,736 462,961
V4RPE488 WCHURCHILL 204374 B 1200 2019-01-04 00:39:00 1:52:24.3 37.5 11 693,732,989 2,044,134
PRD M3SRVADM 566178 B 3939 2019-01-04 07:34:15 ? 1:39:12.3 10.1 35 64,576,151 0
File Analysis Summary Job V4RPE488 WCHURCHILL 204374 on 2019-01-04
ODP Overview
File Name Totals
Call Stack
Details per Itv.
Prompt for text
File Statistics
Library name
File name
Member name
File nbr.
File type
Option
Number of writes
Number of reads
Number of other I/Os
Nbr.of Itvs.
Diff. RRNs
Reuse count
% of Itvs.
RRN span (High-low)
Potentially superfl. I/Os
PRODDATA MMS2RDET MMS2RDET 2 PF I 250,792,585 1,184 263,652,134
PRODDATA MMS2RVIT MMS2RVIT 5 PF I 250,792,549 1,184 79 1,108 94 1,665 250,790,884
PRODDATA MMS2TOTA
PRODDATA MMS2ERKL
PRODDATA MMS2MJU7
PRODDATA MMS2NUMB
PRODDATA MMS2SEQU
File Statistics for Job V4RPE488 WCHURCHILL 204374 on 2019-01-04
ODP Overview
File Analysis
Call Stack
Details per Itv.
Prompt for text
Time
File number
Library name
File name
Member name
File type
Option
Number of writes
Number of reads
Number of other I/Os
Relative record nbr
Share Count
05:30:45 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 242,990,646 0 699 1
05:31:01 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 243,224,530 0 659 1
05:31:16 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 243,455,332 0 213 1
05:31:30 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 243,675,654 0 1,214 1
05:31:45 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 243,914,561 0 1,627 1
05:32:00 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 244,155,640 0 1,627 1
05:32:15 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 244,386,870 0 163 1
05:32:31 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 244,618,817 0 251 1
05:32:45 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 244,831,249 0 826 1
05:33:00 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 245,066,736 0 251 1
05:33:15 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 245,313,338 0 1,627 1
05:33:30 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 245,553,972 0 1,403 1
05:33:45 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 245,804,968 0 1,627 1
05:34:01 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 246,038,634 0 663 1
05:34:15 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 246,272,339 0 1,167 1
05:34:30 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 246,487,713 0 1,144 1
05:34:45 5 PRODDATA MMS2RVIT MMS2RVIT PF I 0 246,705,652 0 251 1

Finding Read Optimization Candidates

Obtaining file/table I/O-statistics → Totals since last IPL:

```
DSPFD LIBNAME/*ALL OUTPUT(*OUTFILE)
```

Significant improvement may be possible if a file/table shows

1. Millions of reads by relatively few different programs,
2. Not too many different records accessed (What is "too many"?)
3. Relatively few (simultaneous) updates / writes (What is "few"?)

   (a trigger program could keep a User Index or User Space updated).

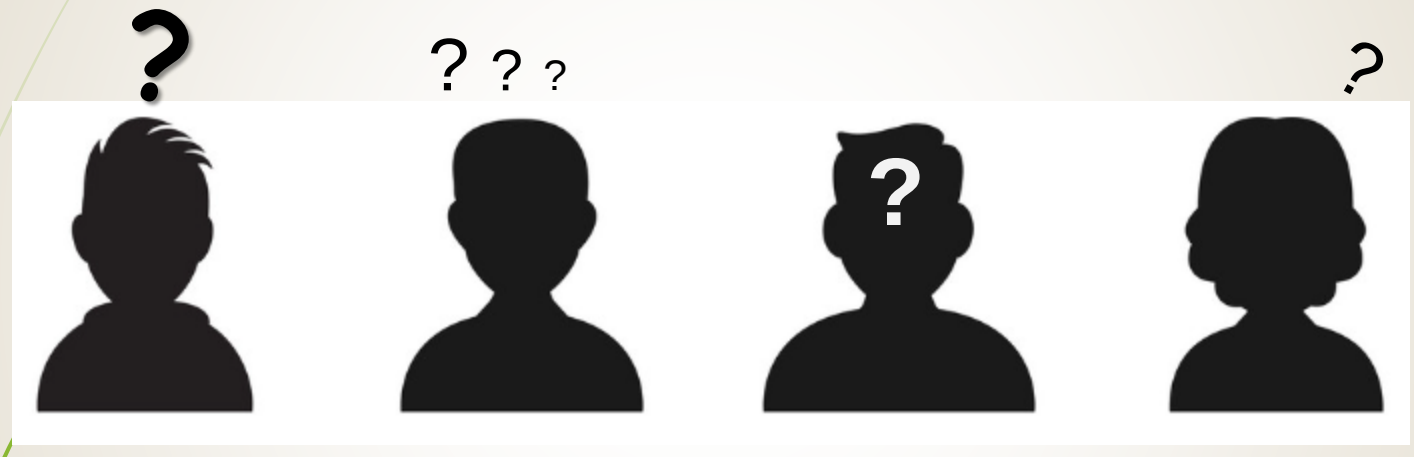
To show job I/O statistics before files are closed:

```
DSPJOB OUTPUT(*PRINT) OPTION(*OPNF)
```

Convert tables/files rarely changed and used by most applications to User Indexes!

E.g. zip codes, prices, sales district table, country codes, most parameters.

Thank you for your attention !



Questions ??

Short video introduction to GiAPA:
https://www.giapa.com/en/product-intro/new-giapa-video/video/giapa

GiAPA®
by iPerformance

→ → → **Free "Performance X-ray" offer on www.giapa.com !** ← ← ←