Version 24, 14 nov 2023 12:44

Tales of the Secret Bunker 2023

research from the parmington foundation

Gus Bjorklund,
head groundskeeper
the parmington foundation

Mike Furgal,
assistant to the head groundskeeper
the parmington foundation

Tom Bascom,
curmudgeon

tpf

PUG CHALLENGE AMERICAS

Burlington, MA, USA
12 – 15 nov 2023

Abstract

The "Secret Bunker" has been used for many interesting OpenEdge RDBMS investigations in past years. Please join your intrepid explorers for another trip into the bowels of the (recently relocated) bunker.

As databases are used and modified over an extended period, database administrators will gain experience and knowledge of what they should have done. They will discover that various improvements in database organization could be made to improve day-to-day performance and maintenance operations. In this talk we will examine some techniques you might use for "online table dump and load" operations with the database.

OpenEdge Release 12 has a variety of new features and capabilities that can make such maintenance tasks more efficient and less disruptive to normal production operations. In the bunker we test some of these capabilities and now we report the results.

About the Secret Bunker

Established in 2002 to investigate Progress on Linux

The Founders

- Foreman
- Bjorklund
- Harlow



2002 Secret Bunker was at this secret location

The Benchmark Laboratory

Secret Bunker 2023

This Year's Secret Bunker Location


BUNKERMUSEUM
JANSJE SCHONG

This Year’s Topic: Dump and Load Optimization

- Test out the new ”proutil <dbname> -C tablereorg
- How does it work ?
- Can it be run online effectively ?
- What can it do for you ?
- Benchmarks and results

2023 Test Machine

Dell PowerEdge 2050 iii (circa 2009)

- 2 XEON E5450 3 GHz 4-core processors
- 64 GB ECC RAM
- PERC/6 RAID controller
- 6 Hitachi 2 TB SATA disk drives
  - /boot (xfs), /home (xfs) and swap on 4x2 TB sata drives in RAID 0.
  - /bi (ext3) on 2 tb drive
  - /ai (ext3) on 2 tb drive

2023 Test Machine Cost

Item	Qty	Cost
Computer	1	$189.00
64 GB ECC RAM	1	$50.00
Disk drives	6	$300.00 (@ $50 ea)
Drive mounting screws	24	$6.32 for 50
SAS Drive carriers	6	$29.94 (@ $4.99 ea)
SAS to SATA adapters	6	$30.00 (@ $ 5.00 ea)
Operating system	1	$0.00
Total	44	$605.26

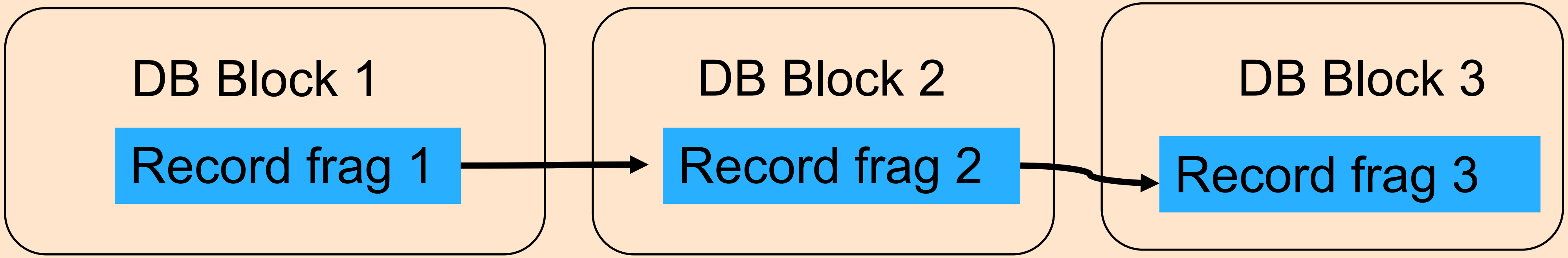
"bigrow test" time: 1.1 seconds. (82.27 mb/sec)

tpf

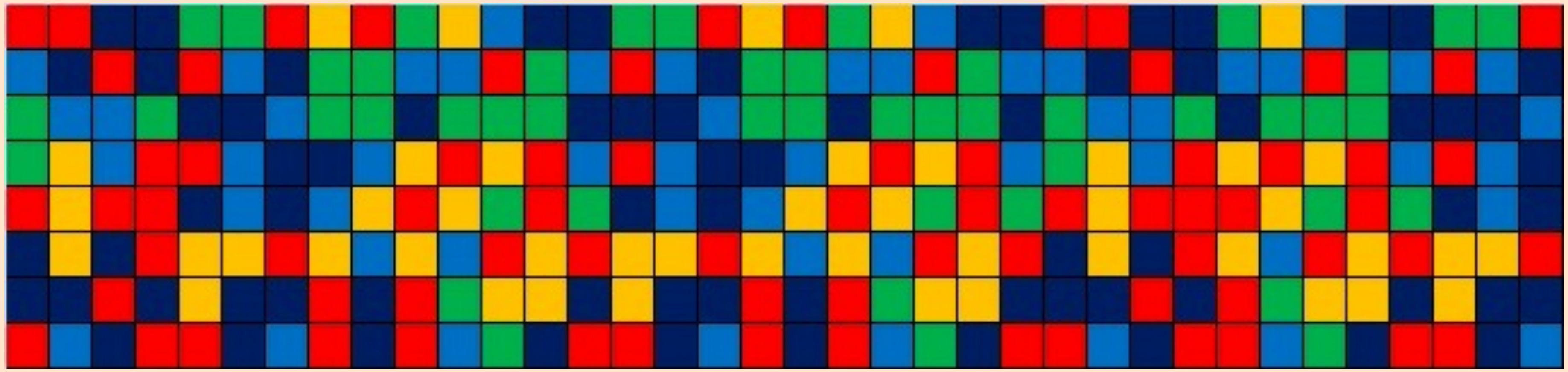
About Scatter and Fragmentation

Database Scatter

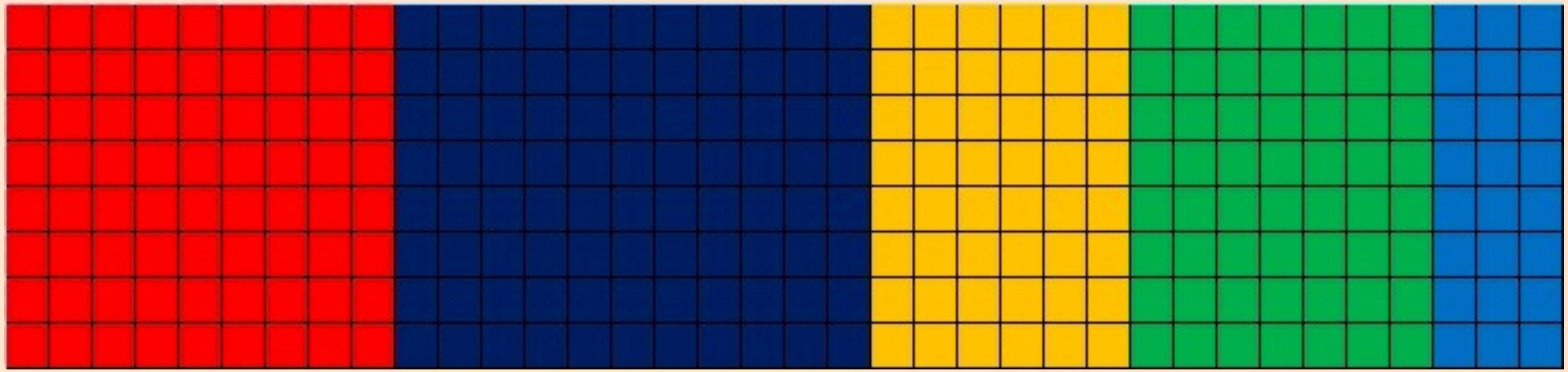
- Comes in 3 forms
  - Record Fragmentation
  - Physical Scatter
  - Logical Scatter
- Record Fragmentation



Physical Scatter



not good



good

each color represent a different table

Logical Scatter

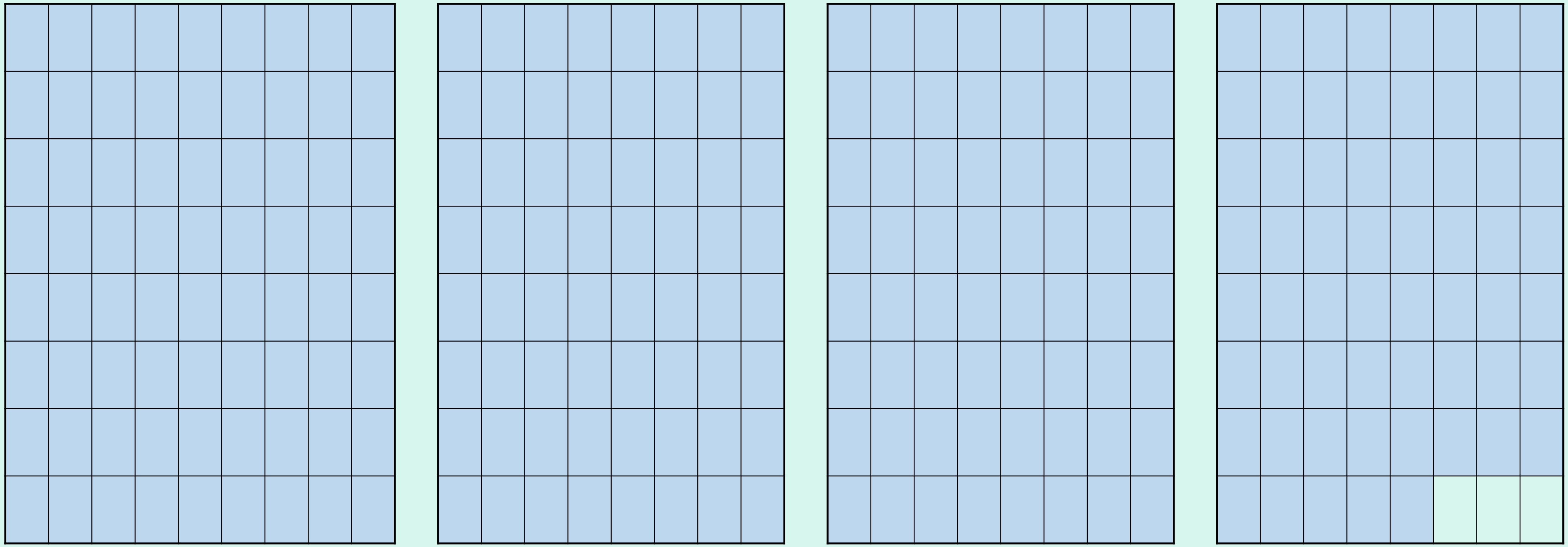
- How ordered are the records by some index?
- Each index gives a different logical ordering
- Are rowids in the same order as the keys?
  (there can be only one physical ordering)

1	250	167	40	407	424	177	124	456	321	286	46	37	390	244	259	202	135	336	359	480	108	83	274	462	394	224	30	341	441	451	95	51	316	172	104	32	374	150	323
219	282	436	23	62	400	75	427	473	458	67	41	125	326	198	309	152	213	284	255	224	65	385	267	248	337	439	147	383	460	364	474	419	39	170	465	405	154	252	80
414	191	268	197	144	411	164	84	130	60	123	461	377	307	272	11	221	450	76	351	3	109	290	128	380	118	429	74	345	116	63	121	18	357	468	106	420	434	295	155
475	301	141	22	397	437	183	300	449	218	233	347	261	187	196	9	143	146	20	17	24	331	243	303	45	452	192	269	79	256	422	239	199	7	367	356	153	432	446	56
157	131	194	312	457	242	349	102	348	190	49	260	201	245	291	158	85	204	388	416	319	174	181	279	391	6	220	113	280	225	58	138	229	477	313	373	200	103	69	217
235	230	15	289	276	72	188	264	50	298	61	402	184	318	470	71	270	241	119	127	163	148	360	262	297	142	251	165	314	384	173	294	305	315	342	13	471	363	186	472
35	410	403	404	296	479	110	333	185	476	382	33	401	317	205	379	418	93	231	240	73	285	171	136	425	57	226	206	393	166	232	114	408	343	339	223	160	370	31	381
101	389	327	392	21	212	409	358	145	53	406	299	426	365	362	78	448	350	98	92	322	175	54	90	96	354	70	105	431	126	249	273	43	34	330	36	328	338	447	361
412	454	100	2	320	423	26	210	387	263	162	208	139	292	19	428	247	87	52	463	209	133	430	455	180	195	398	266	111	134	467	435	372	308	156	310	168	469	169	369
5	253	81	112	48	258	215	352	44	4	288	86	120	466	47	12	478	443	395	216	324	55	340	203	334	287	332	417	355	413	386	129	64	91	459	438	16	265	464	97
227	42	346	325	421	44	306	182	277	238	302	179	371	376	214	82	77	8	353	28	246	396	207	211	433	378	14	271	25	140	335	68	445	151	375	59	161	283	88	329
453	122	38	293	415	189	99	159	137	193	117	27	237	115	94	236	222	304	178	89	176	281	440	29	278	254	275	10	107	66	368	344	442	366	149	311	257	399	132	228

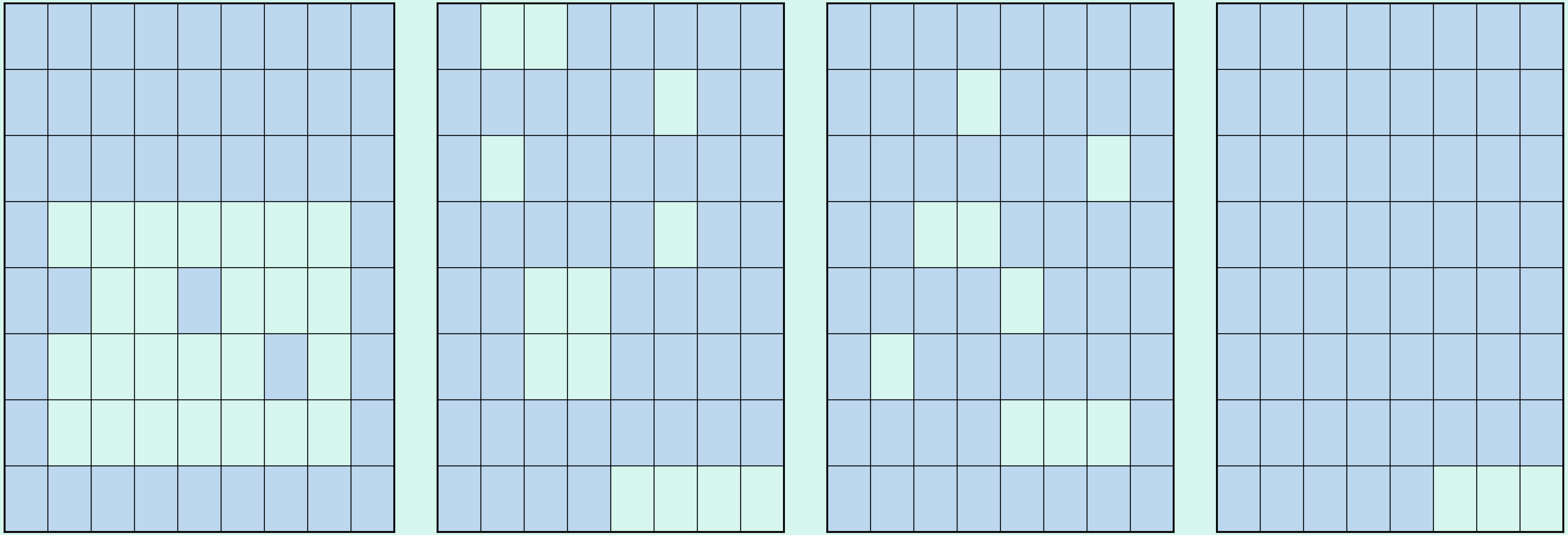
Tablereorg

- An online utility that puts rows into a physical order that matches order of one index
- Operates on a table (or part of table) at a time
  - Removes Record Fragmentation (where possible)
  - Removes Physical Scatter
  - Removes Logical Scatter

A table with 4 data Clusters (type ii data area)

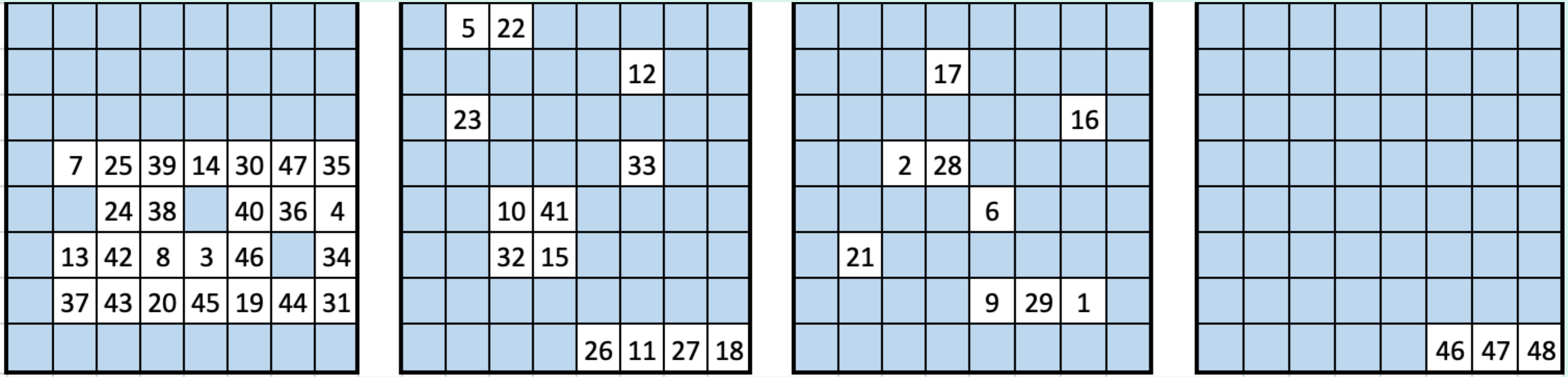


Holes after deleting some records



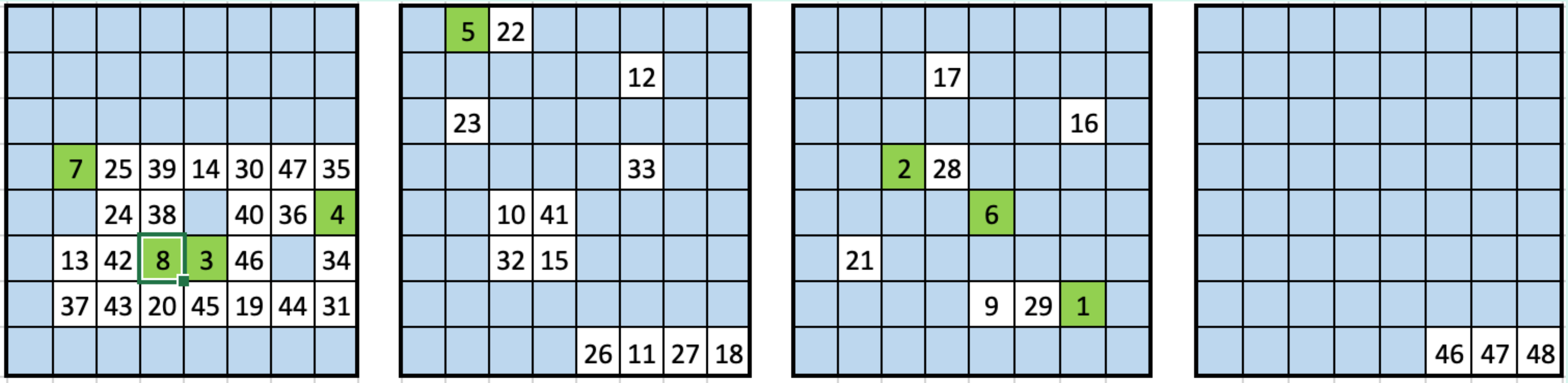
RM chain after deleting records

RM chain is a list of blocks that have usable space for additional records



Numbers represent the order of the RM chain

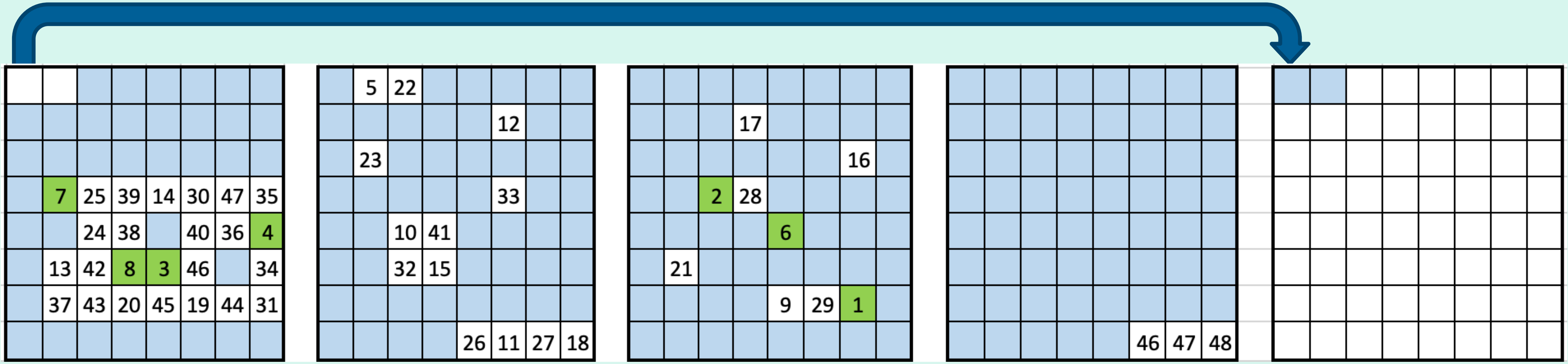
Green spaces are where new records go



Tablereorg

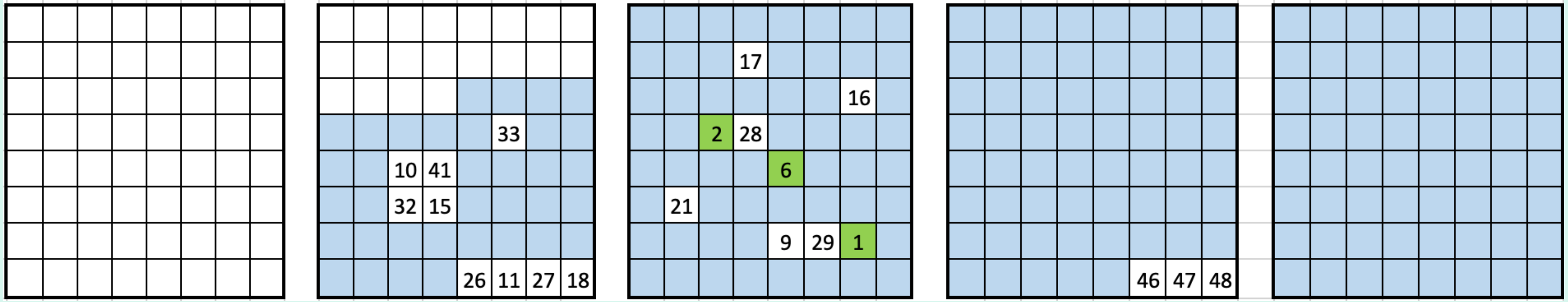
tablereorg moves records to establish a NEW physical ordering
to match order of index you specify

Record one, two, etc move to a new cluster



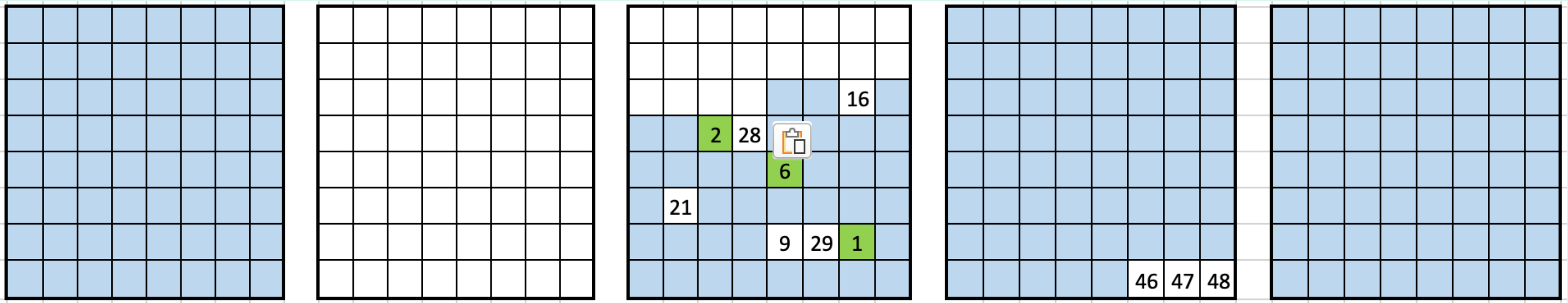
Tablereorg

New cluster fills, old cluster available for reuse

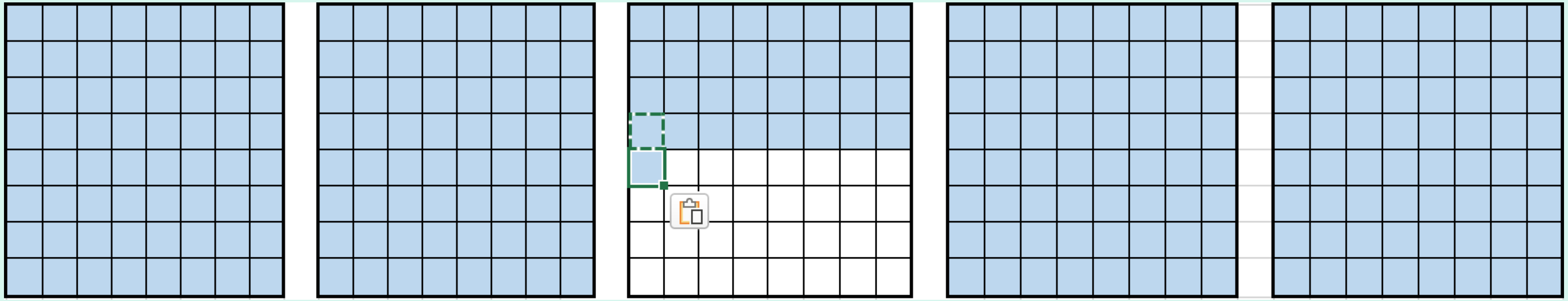


Tablereorg

Available clusters are reused



When completed there are no "holes"



Now let's try it out

Created 6 scenarios with

- ATM Database
  - Accounts Table
  - 10,000,000 records
  - 1,000,000,000 records
  - 1,000,000,000 records varying size from 100 to 5000 bytes
  - 500,000,000 records
- Demo database
  - Customer table
  - 33 records

Scenario 0: starting conditions

- Regular ATM database but small
  (only 10,000,000 account records)
- Unused after building
- All records a little over 100 bytes long
- Database is about 1.5 GB
  - 371,020 blocks   (4k)
- No fragmented records
- 94% RM space utilization
- No free clusters
- 3 blocks on account RM chain

Scenario 0: table reorg 1

- proutil atm -C tablereorg account recs 1000
- Took 6 min 57 sec.
- Database grew from 1.5 GB to 3.1 GB
  - From 371,020 to 574,668 blocks (4k)
- No fragmented records
- 60% RM space utilization
- No free clusters
- 328,100 blocks on the RM chain

Scenario 0: table reorg 2

- reorg of just reorg'ed database
- proutil atm -C tablereorg account recs 1000
- Took a few seconds longer than first time (7 min +)
- Database grew slightly from 3.1 GB to 4.2 GB
  - From 574,668 to 593,676 blocks (4k)
- No fragmented records
- 62% RM space utilization !!!!
- No free clusters
- 531,596 blocks on the RM chain
- Records were moved but nothing much was accomplished

Scenario 1: Same as previous but bigger (100,000,000 accounts)

- Database size is now 14 GB
- Again unused after building
- There are 3,752,908 blocks
- No free clusters
- 2 blocks on the RM chain
- Zero fragmentation

Scenario 1: table reorg 1

- proutil atm -C tablereorg account recs 1000
- Took 72 min 54 sec.
- Database grew from 14 GB to 22.2 GB
  - From 3,752,908 to 5,815,500 blocks (4k)
- No fragmented records
- 60% RM space utilization
- No free clusters
- 3,332,660 blocks on the RM chain

Scenario 1: table reorg 2

- proutil atm -C tablereorg account recs 1000
- Took 67 min 12 sec.
- Database grew from 22.2 GB to 22.9 GB
  - From 5,815,500 to 5,998,028 blocks (4k)
- No fragmented records
- 61.2% RM space utilization
- No free clusters
- 5,395,252 blocks on the RM chain

Scenario 1: table reorg 3

- No change in database size
- 16 additional blocks on RM chain
- Nothing much happened
  - records moved but otherwise result was the same

Scenario 2: Make account records vary in size

- 100,000,000 accounts
- Record size varies from ~100 to ~500 bytes
- 1 in 20 records is ~100 to ~1500 bytes
- Database size is 9,916,940 blocks – 37 GB
- RM chain has 2,021,146 blocks
- There are 190,872,537 fragments
- 95.3 % RM space utilization

Scenario 2: table reorg 1

- Took 86 min 35 sec
- Database grew from 37 GB to 53 GB
- From 9,916,940 to 13,003,212 blocks
- Zero fragmentation
- 71.84% RM space utilization
- RM chain has gone from 2,021,146 to 11,262,187 blocks

Scenario 3

- Create an Account record, then create another account record with the account.id x -1.
  - Total 1,000,000,000 records
  - 1, -1, 2, -2, 3, -3, 4, -4, etc
- Delete all negative accounts
  - 1, empty, 2 empty, 3, empty, 4, empty, etc
  - Total 500,000,000 records

Scenario 4

- Create an Account record, assign the account.id x -1
- Once complete, create Account records as normal
- Result
  - Total 1,000,000,000 records
  - -1,-2,-3, …. -499,999,999, -500,000,000,1,2,3,4, etc
- Delete all negative accounts
  - Empty, empty, ….., 1,2,3,4, etc
  - Total 500,000,000 records

Warning: Rathole ahead

How can you delete 500 MM records quickly?

Simplest way to delete 500 MM records:

```
for each account:
    delete account.
end.
```

Google leads us to a better way

Progress community article P36834 says to do this:

```
define var recNum as int no-undo.
define var tranSize as int no-undo initial 10000.

recNum = 0.
outer:
do while true transaction:
    for each account exclusive-lock:
        recNum = recNum + 1.
        delete account.
        if ((recNum modulo tranSize) eq 0) then next outer.
    end.
    leave.
end.
```

Progress community advice

Instead of 500 MM transactions
50,000 transactions.

This worked great --- until it didn't.
UNIQUE index on account.id
caused deleted index entry place-holders.

Eventually minutes went by after
every 10,000 deletes to skip over
these place-holders.

FOR EACH account:
DELETE account.
END.

500 MM transactions.
This worked best.

FOR EACH account:
DELETE account.
END.

500 MM transactions.
This worked best.

Here, 1 FOR EACH.
The 10,000 records / tx code
does 50,000 FOR EACH.

Rathole concluded

Scenario 3 – Interleaved deleted rows

- Tablereorg with all defaults
  - proutil atm –C tablereorg account
  - Took 10:52:03 to complete

- Tablereorg with 1,000 records per transaction
  - proutil atm –C tablereorg account recs 1000
  - Took 10:40:18 to complete

Scenario 3 - Result

- The database size grew from 126 GB to 171 GB
- Why? Let's investigate
  - Accounts Area
    - Records Per Block: 64
    - Data Cluster Size: 512
  - Records: 500,000,000
  - Average size: 124 bytes
  - Database Blocksize: 8192
  - How many records fit in a block?
    - 8192 – (128ish) = 8000ish / 124 bytes = 64 records should fit in a block

Scenario 3

- A freshly loaded database has an accounts area of 65,007,517,696 bytes
  - 512 8K blocks per cluster = 15,499 clusters.
- The database prior to the table reorg has an accounts area of 130,036,006,912 bytes
  - 512 8K blocks per cluster = 31,003 clusters.
- The tablereorg database has an accounts area of 178,115,313,664 bytes
  - 512 8K blocks per cluster = 42,466 clusters.
- Tablereorg added 11,463 Data Clusters

QUIZ !

Why are there

11,463 additional clusters

after the reorg?

2 Issues Found:

Area Size Growth

2 Hour Startup Time

Let’s Try Scenario 4

A large chunk of empty space followed by contiguous records

Scenario 4 – Large chunk of free space

- Time to complete with 1,000 rows per transaction
  - 7:47:44
- Database Size before Reorg:
  - 126 GB
- Database size after Reorg:
  - 126 GB
- Account Area 130,036,006,912 bytes
  - 512 8K blocks per cluster = 31,003 clusters.
- No additional Data Clusters added this time

What about the startup time?

Proutil db –C tablereorg arguments

proutil *db-name* -C tablereorg **[***owner-name.***]** *table-name*
[info **] |**
[[resume **|** resume-numrecs *n* **|** resume rowid *n***]**
[nosmartscan **]**
[restrict **[** EQ *value* **] |**
[LT **|** LE *high-value* **] |**
[GT **|** GE *low-value* **[** AND LT **|** LE *high-value* **]]]**
[useindex *index-name* **] [** recs *n* **]**
[searchdepth *n* **]**
[reusepercent *n* **]]**
[tenant *tenant-name* **|** group *group-name* **|**
partition *partition-name* **|** composite initial **]**

Startup time solved

searchdepth

Specifies the percentage of the record free chain to search for free blocks to be used during the table reorganization.
The minimum is 0, the maximum is 100, and the default is 100.

- proutil atm -C tablereorg account recs 1000 searchdepth 5
  - Since the record free chain (RM chain) was large it spent all the startup time scanning
  - Before table reorg, there were 31,003 Data Clusters.
  - A fresh Dump/Load database has 15,499 Data Clusters.
  - This leaves 15,504 empty clusters, or 7,938,048 blocks to scan

Scenario 4: What about Performance Impact?

Modified ATM benchmark

- 150 Users
- 7 Users doing updates as normal ATM activity
  - Update Account, Teller, and Branch Balance
  - Create History Record
- 143 Users doing reading
  - Randomly Read 101 Account Record
  - Read a Teller and Branch Record

ATM Results

Baseline with 150 users

Trans Per Sec	Records Read	Records Updated	Longest Wait
258.00	68,878.00	8,381.00	4.30
184.60	45,849.00	9,383.00	4.20
176.10	43,032.00	9,657.00	4.10
174.60	43,240.00	8,988.00	3.90

During Tablereorg

Trans Per Sec	Records Read	Records Updated	Longest Wait
141.90	38,072.00	4,341.00	146.60
135.30	37,667.00	2,786.00	27.30
135.70	38,483.00	2,081.00	72.30
139.40	39,106.00	2,555.00	103.00

Final Test – Compared to a traditional Dump/Load

- proutil atm -C tablereorg account recs 1000 searchdepth 5
  - 8 hours, 40 minutes to complete
- Binary Dump/Load
  - Dump: 43 minutes
  - Load 24 minutes
  - Index Rebuild: 8 minutes
  - Total Time: 1 hour, 15 minutes
- Binary Dump/Load
  - Must disable AI, then re-enable
  - Must disable OE Replication, then re-enable and rebaseline
- Tablereorg allows AI and OE Replication to stay intact

Comparison of 3 different methods

500,033 customer records from the demo database

	Time	AI Size	AI Notes	Avg Note Size
Partitioning	2:07	2,263,482,368	10,543,724	214.68
Table Reorg	1:25	2,063,204,352	7,643,092	269.94
Buffer copy	2:00	2,123,497,472	9,043,655	234.81
Dump/Load	0:46	N/A	N/A	N/A

Lastly, something a bit different!

Scenario 5: A Homework Exercise !

Try this at home:

```
mkdir homework
cd homework
prodb foo isports
echo a foo.a1 >addai.st
prostrct add foo addai.st
probkup foo /dev/null
rfutil foo -C aimage begin
proutil foo -C dbanalys >foo.ana1
proutil foo -C tablereorg customer recs 1000
proutil foo -C dbanalys >foo.ana2
diff -Bw foo.ana1 foo.ana2
rfutil foo -a foo.a1 -C aimage scan verbose >aiscan.txt
more aiscan.txt
```

So what can tablereorg do for you?

Tablereorg will attempt to

- Minimize record fragmentation
- Make physical order align with index key order
  - For one index you choose
  - Rowid order and key order will be the same (as much as feasible)
  - Choose wisely (default is primary key)
- Performance will improve for some portions of the application
- Operates on one table at a time

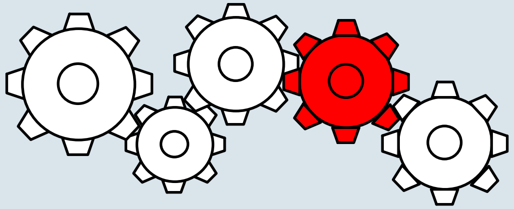
Tablereorg will also

- Move all records
  - ROWID's will change
  - All indexes for the table will be updated
- Table's storage area will expand (if not enough free clusters)
  - Previous record locations become free RM space
  - Might fail if data area fills or reaches max size
- May expand RM chain
  - Depending on
    - Available free clusters
    - Existing RM chain
    - Searchdepth setting

Conclusions

- Tablereog is a useful tool to remove fragmentation and scatter
- Appears to be pretty fast
- The database may increase in size depending on how contiguous free space is organized
- Should be run at off hours when the user load is not high
- TEST before using in production !!!
- Can be restarted if something goes wrong
- Be careful of the defaults to limit startup time
- As always YMMV
  - Transportation, meals, and accommodation not included

Questions



?

research from the parmington foundation