DBG Software

Real Performance…It's About Time

SQL Performance

Data Sequencing

Data Sequencing Highlights

- ♦ A sequenced table will attempt to store it's rows sorted in order by a specified column or columns
- ♦ Sequencing tables that are usually accessed sequentially by the sorting strategy is good practice because fewer I/O requests (logical or physical) are required to retrieve rows
- ♦ Sequencing can result in significant performance gains across the board (CPU, I/O, Elapsed Time) by not only retrieving the row you need, but also the rows you'll need next

What Is Data Sequencing?

On some DBMS's, data sequencing is enforced by a particular index used to represent the desired collating sequence for the data in that table. There can be only one collating sequence per table, since physically the data can be stored in only one order.

Depending on the DBMS, the data may not always be physically maintained in exact collating sequence. When a sorting strategy has been defined for a table, the DBMS will act in one of two ways:

1. When new rows are inserted, the DBMS will physically maneuver data rows and pages to fit the new rows into the defined collating sequence. For example in Oracle, this is known as an Index Organized Table (IOT).

(or)

2. When new rows are inserted, the DBMS will try to place the data into the defined collating sequence, but if space is not available on the required page the data may be placed elsewhere. In this case, data may become un-sequenced over time and require reorganization. In Oracle, the metadata column to monitor for data cluster health is in the DBA_INDEXES table and it's called CLUSTERING_FACTOR. As CLUSTERING_FACTOR approaches the number of table blocks the table is well sequenced with regards to the definition of that index. As CLUSTERING_FACTOR approaches the number of table rows the table is poorly sequenced with regards to the definition of that index.

When Do You Sequence Data?

- Join columns, to optimize SQL joins where multiple rows match for one or both tables participating in the join
- Foreign key columns because they are frequently involved in joins and the DBMS accesses foreign key values during declarative referential integrity checking
- When the most critical queries retrieve ranges of data
- When data is frequently sorted together (JOIN's, ORDER BY, GROUP BY, UNION, SELECT DISTINCT)

Data Sequencing Considerations

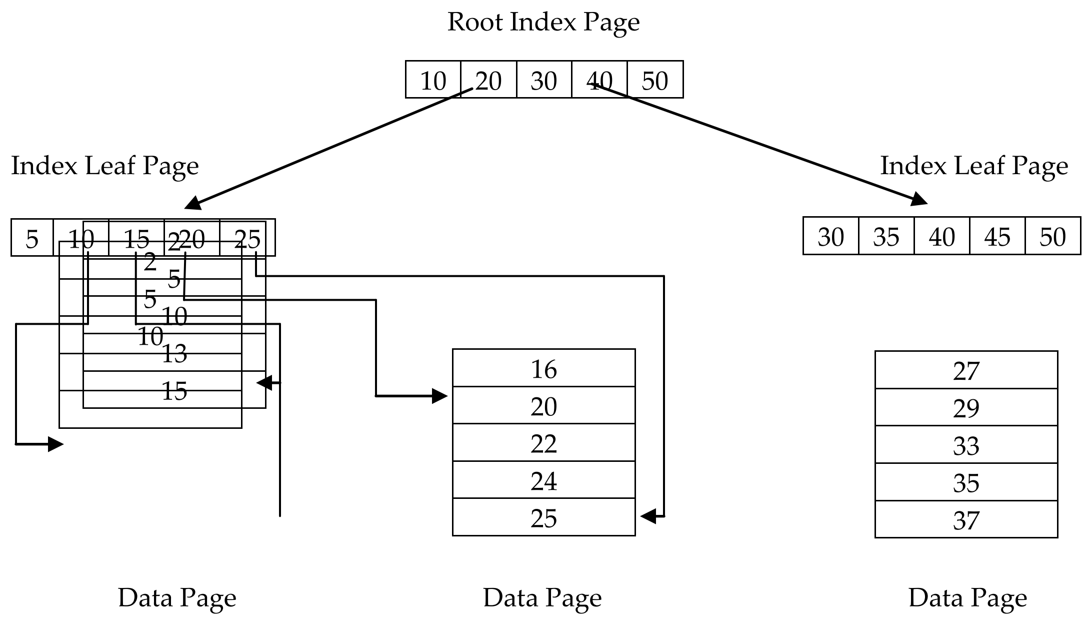
- Before sequencing a table, consider the frequency of modifications. This will help determine how often reorganizations to re-sequence the data will need to occur.
- When a table has multiple candidates for sequencing, weigh the cost of sorting against the performance gained by sequencing for each candidate key.
- As a general rule of thumb, it is usually a good practice to define a sequencing strategy for each table that is created (unless the table is very small).
- Data sequencing is generally not recommended for primary key columns because the primary key is, by definition, unique. However, if ranges of rows frequently are selected and ordered by primary key values, a sequencing strategy may be beneficial.

Data Sequencing Illustrated

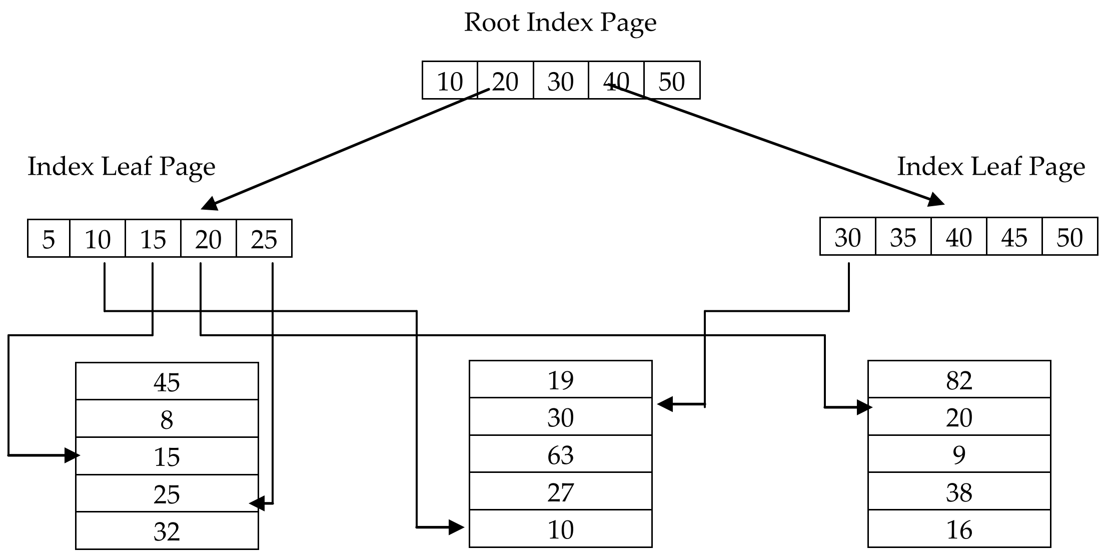
The illustration below demonstrates the difference between sequenced and un-sequenced data and indexes; the sequenced index is on top, the un-sequenced index is on the bottom. Notice, the entries on the leaf pages of the top index are in sequential order in the data pages. In other words, they are sequenced.

(continued)

Sequenced



Un-Sequenced



Also notice in the above examples, that sequenced indexed access as it progresses from leaf page to leaf page will not have to open the same data page twice. Un-sequenced index access, however, will open the same data page multiple times because the data is spread throughout the table.

What Is The Cost Of A Logical Read?

One could think the benefits of data sequencing only apply to physical I/O operations and that if data is kept in memory, logical I/O operations would not see benefit. However, even when data is being accessed in memory there is overhead associated with every logical I/O. If data is un-sequenced in a physical block, it will also be un-sequenced in a corresponding memory buffer and the same logical I/O inefficiencies would apply as compared to physical I/O.

First, it's important to note several items that affect the number and overhead associated with logical I/O reads:

1. Whether data sequencing is implemented for sequential access
2. Quality of the SQL execution plan
3. Fetch array size
4. Whether table-prefetching is enabled

Focusing just on data sequencing, if the rows needed are sequenced together in a block (or series of blocks) Oracle will open that memory buffer once and read multiple rows applicable to the current SQL.

If the rows are not sequenced together, several items should be considered that would represent the overhead of a logical I/O read (this list may not be all inclusive):

1. Oracle calculates the hash value of the needed block and searches the SGA for where it is located
2. Oracle checks the System Change Number (SCN) of the block and compares it with the SCN of the current transaction.
3. If the instance is part of a RAC setup, the directory is checked and it is ascertained whether any other instance has modified the block.
4. Then, the block is paged in the address space of the requesting process

Regarding read consistency in steps #2 & #3 above, further overhead would occur because the SCN would come into play and the consistent version of the block must be constructed from the undo segments.

The four overhead items above represent a chain of events that occurs during a logical I/O read. Before that chain is traversed, a latch is obtained on the chain. If the chain is hot, the closer a system gets to processor saturation the more troublesome this latching becomes and this can be a problem. Looking at sleeps on cache buffer chains can indicate a serious problem. In other words, referencing un-sequenced data in memory in a sequential manner means much higher CPU resource usage.

In summary, considering the overhead of a logical I/O read, reducing that overhead as much as possible seems a pertinent strategy. Data sequencing would reduce the amount of logical I/O overhead by allowing a memory buffer to be opened once and read from multiple times, thus incurring only one iteration of the overhead discussed above.

How Is Data Sequencing Done?

Performing data sequencing can be accomplished in either an online or offline fashion. Online meaning the application remains up and running during the reorganization, offline meaning the application must be brought down.

Depending on the application, performing the sequencing in an online fashion can be desirable. For example, Database Brothers Inc. (DBI) has a product called "Brother Wolf" that will perform an online reorganization and gives the option to also sequence the data by an index.

A datasheet on this product can be found at http://www.dbisoftware.com/brother-wolf.php

Data Sequencing Examples

Example #1

A 10,000 row unsequenced test table was created in Oracle and populated in the following fashion:

```
create table UNSEQUENCED_data(row_number int, last_name char(10), last_column
char(1000));
create index UNSEQUENCED_index on UNSEQUENCED_data (last_name);
begin
for i in 1..10000 loop
insert into UNSEQUENCED_data values (i, 'JONES',' ');
end loop;
end;
analyze index UNSEQUENCED_index compute statistics;
```

A series of updates were run to change the last name of "JONES" to "SMITH" for 298 of the records (spread throughout the table). Here is an example of one of the UPDATE's:

```
UPDATE UNSEQUENCED_DATA SET LAST_NAME = 'SMITH' WHERE ROW_NUMBER = 100;

select count(*) from UNSEQUENCED_data where last_name = 'SMITH';
  COUNT(*)
----------
       298
```

A sequenced version of the same table was then created:

```
create table SEQUENCED_data(row_number int, last_name char(10), last_column char(1000));
create index SEQUENCED_index on SEQUENCED_data (last_name);
insert into SEQUENCED_data
select *
  from UNSEQUENCED_data
 order by last_name
/
analyze table SEQUENCED_data compute statistics;
analyze index SEQUENCED_index compute statistics;

select count(*) from SEQUENCED_data where last_name = 'SMITH';
  COUNT(*)
----------
       298
```

Here are some performance metrics comparing executions against both of the tables using the following SQL:

```
select *
  from [table]
where last_name = 'SMITH'
/
```

This is the runtime execution plan for the unsequenced SQL:

```
--@@  RunTime Plan

  0        SELECT STATEMENT   (Cost=781)
  1   0      TABLE ACCESS BY INDEX ROWID UNSEQUENCED_DATA (Cost=781)
  2   1        INDEX RANGE SCAN UNSEQUENCED_INDEX (Cost=16 Columns=1)
```

This is the runtime execution plan for the sequenced SQL:

```
--@@  RunTime Plan

  0        SELECT STATEMENT   (Cost=738)
  1   0      TABLE ACCESS BY INDEX ROWID SEQUENCED_DATA (Cost=738)
  2   1        INDEX RANGE SCAN SEQUENCED_INDEX (Cost=23 Columns=1)
```

	Executions	CPU	Logical Reads	Physical Reads	Elapsed Time	Rows Returned
UNSEQUENCED_Data						
First Run Stats	1	0	320	300	1.238579	298
Total Stats	10	0.01	3200	300	1.264901	2980
Average Stats	10	0.001	320	30	0.1264901	298
SEQUENCED_Data						
First Run Stats	1	0	83	3	0.035418	298
Total Stats	10	0	830	3	0.051661	2980
Average Stats	10	0	83	0.3	0.0051661	298
Percentage Gain						
First Run Stats		n/a	74.06%	99.00%	97.14%	
Total Stats		100.00%	74.06%	99.00%	95.92%	
Average Stats		100.00%	74.06%	99.00%	95.92%	
"In Memory" Stats (ie, w/o first run)		100.00%	74.06%	n/a	38.29%	

Example #2

For the following actual Oracle tables for a mission critical application at a fortune 500 company, the RENT_CNTRCT & DRVR tables were copied and sequenced with the following as sequencing sequences:

```
RENT_CNTRCT: PCKUP_IORG_ID then RENT_CNTRCT_STAT_CDE then DRVR1_LNAM
       DRVR: PCKUP_IORG_ID then LNAM

SELECT /*+ FIRST_ROWS */
       DISTINCT rent_cntrct.rent_cntrct_nbr AS rc_nbr, gui_resv_nbr, lgcy_resv_nbr,
resv_orig_cde
     , NVL2(pckup_tim, TO_DATE (TO_CHAR (pckup_dte, 'MM/DD/YYYY'), 'MM/DD/YYYY'),
pckup_dte ) AS ticket_dte
     , NVL2 (pckup_tim, (TO_NUMBER (SUBSTR (pckup_tim, 1, 2)) * 3600000 ) + (TO_NUMBER
(SUBSTR (pckup_tim, 4, 2)) * 60000), NULL ) AS ticket_tim
     , NVL2 (rtn_tim, TO_DATE (TO_CHAR (rtn_dte, 'MM/DD/YYYY'), 'MM/DD/YYYY'), rtn_dte )
AS return_dte
     , NVL2 (rtn_tim, (TO_NUMBER (SUBSTR (rtn_tim, 1, 2)) * 3600000) + (TO_NUMBER (SUBSTR
(rtn_tim, 4, 2)) * 60000), NULL ) AS return_tim
     , NVL2 (rtn_tim, rtn_dte, rtn_dte + 2) AS return_date_sort
     , rent_cntrct_stat_cde, rent_cntrct.pckup_iorg_id AS rc_iorg_id,    lnam, fnam
  FROM [rent_cntrct table]
     , [drvr table]
  WHERE drvr.rent_cntrct_nbr = rent_cntrct.rent_cntrct_nbr
   AND drvr.drvr_id = 1
   AND rent_cntrct.pckup_iorg_id = 11854
   AND rent_cntrct.rent_cntrct_stat_cde IN (5)
   AND rent_cntrct.drvr1_lnam LIKE 'SMI%'
 ORDER BY lnam, fnam
/
```

This is the runtime execution plan for the unsequenced SQL:

```
--@@  RunTime Plan

  0        SELECT STATEMENT   (Cost=20)
  1   0      SORT UNIQUE  (Cost=13)
  2   1        NESTED LOOPS   (Cost=6)
  3   2          TABLE ACCESS BY INDEX ROWID RENT_CNTRCT (Cost=4)
  4   3            INDEX RANGE SCAN XIE20RENT_CNTRCT (Cost=3 Columns=2)
  5   2          TABLE ACCESS BY INDEX ROWID DRVR (Cost=2)
  6   5            INDEX UNIQUE SCAN XPKDRVR (Cost=1 Columns=2)
```

Column breakdown for the above index usage:

```
XIE20RENT_CNTRCT                    PCKUP_IORG_ID
                                    DRVR1_LNAM
                                    RENT_CNTRCT_NBR

XPKDRVR                             RENT_CNTRCT_NBR
```

```
                                   DRVR_ID
```

This is the runtime execution plan for the sequenced SQL:

```
--@@  RunTime Plan

  0         SELECT STATEMENT   (Cost=35)
  1   0       SORT UNIQUE  (Cost=28)
  2   1         NESTED LOOPS   (Cost=21)
  3   2           TABLE ACCESS BY INDEX ROWID RENT_CNTRCT_ORGANIZED (Cost=19)
  4   3             INDEX RANGE SCAN NEWJG_XIE20RENT_CNTRCT (Cost=3 Columns=2)
  5   2           TABLE ACCESS BY INDEX ROWID DRVR_ORGANIZED (Cost=2)
  6   5             INDEX UNIQUE SCAN NEWJG_XPKDRVR_ORGANIZED (Cost=1 Columns=2)
```

Column breakdown for the above index usage:

```
NEWJG_XIE20RENT_CNTRCT             PCKUP_IORG_ID
                                   DRVR1_LNAM
                                   RENT_CNTRCT_NBR

NEWJG_XPKDRVR_ORGANIZED            RENT_CNTRCT_NBR
                                   DRVR_ID
```

As was mentioned above, the following collating sequences were used:

```
RENT_CNTRCT: PCKUP_IORG_ID then RENT_CNTRCT_STAT_CDE then DRVR1_LNAM
       DRVR: PCKUP_IORG_ID then LNAM
```

This means the performance gains shown below would be even greater if we would sync up the collating sequences between RENT_CNTRCT & DRVR by either:

1. Denormalizing RENT_CNTRCT_STAT_CDE to DRVR and include it in the sequencing sequence for DRVR

(or)

2. Removing RENT_CNTRCT_STAT_CDE from the sequencing sequence for RENT_CNTRCT

	Executions	CPU	Logical Reads	Physical Reads	Elapsed Time	Rows Returned
Unsequenced: RENT_CNTRCT & DRVR						
First Run Stats	1	0.08	778	714	6.405633	56
Total Stats	10	0.14	7276	714	6.446833	560
Average Stats	10	0.014	727.6	71.4	0.6446833	56
Sequenced: RENT_CNTRCT & DRVR						
First Run Stats	1	0	602	128	0.872788	56
Total Stats	10	0.01	5975	128	0.906199	560
Average Stats	10	0.001	597.5	12.8	0.0906199	56
Percentage Gain						
First Run Stats		100.00%	22.62%	82.07%	86.37%	
Total Stats		92.86%	17.88%	82.07%	85.94%	
Average Stats		92.86%	17.88%	82.07%	85.94%	
"In Memory" Stats (ie, w/o first run)		83.33%	17.31%	n/a	18.91%	

Example #3

To demonstrate the benefits of data sequencing on another mission critical application, and on a different platform (DB2/400), please note the metrics below:

	Unsequenced Data	**Sequenced Data**
Iteration #1		
Open	39.19sec	.27sec
Fetch	17.88sec	.23sec
Total	57.07sec	.50sec
Iteration #2		
Open	3.92sec	.10sec
Fetch	.21sec	.24sec
Total	4.13sec	.34sec

*** The above times are for the following SQL:**

```
Select X4LADT
     , RURFNM
     , RUCUTI
     , RUPHNO
  from A4rprd
     ,
a4xref30
 WHERE X4CUID  = RUCUID
   and X4CUTI  = RUCUTI
   AND X4CUID  = ?
   and X4BCOF  = ?
   and X4STCD  = ?
   and X4LADT <= ?
FOR READ ONLY
```

Just as in the Oracle examples earlier, the only difference between the tests is sequenced data (ie, the execution plans are the same).

After data sequencing was first suggested for this application it was quickly discovered that no DB2/400 third party software vendors had a reorg utility that sequenced data (like Brother Wolf in the Oracle space). However, after sharing the above numbers with a software company called iTera Inc., they included data sequencing as an option in their "Reorg While Active" product and that feature is now marketed on their website.