Tuning guidelines and principles to enhance database performance in WebSphere Commerce

Modifying the default data model to maintain server throughput during peak events in the WebSphere Commerce environment

Noureddine Brahimi (nbrahimi@ca.ibm.com)
Staff Software Developer
IBM

16 December 2014

You can modify the default WebSphere® Commerce data model to maintain optimal system performance during periods of high user connectivity, such as peak shopping events. A heavy workload, simulating a peak shopping event on the original data model, is run and analyzed. Based on that analysis, the data model is reconfigured to optimize database and application performance.

Introduction

Online e-commerce applications tend to become the most stressed during peak shopping events, such as during Black Friday in the United States. Application performance and throughput affect back-end system performance, particularly database performance. A decrease in database performance can lower throughput, impact application performance, and ultimately cause a negative user experience.

By optimizing the database input/output, you can improve performance and concurrency. This improvement enables the application to support higher numbers of e-commerce transactions and maintain a consistent user experience. Changing the default WebSphere Commerce data model maintains throughput without impacting database response time and concurrency. Modifying the data model requires moving tables from their original table spaces to different ones.

Understanding the data model

The default WebSphere Commerce data model comes with three regular table spaces:

1. TAB16K contains 24 tables
2. TAB8K contains 67 tables
3. USERSPACE1 contains 804 tables
IBMDEFAULTBP is the buffer pool that is linked to USERSPACE1. It is also linked to SYSCATSPACE, TEMPSPACE1, SYSTOOLSPACE, and SYSTOOLSTMPSPACE, as shown in the following output:

### Listing 1. Table spaces with corresponding buffer pools

<table>
<thead>
<tr>
<th>TABLE_SPACE</th>
<th>BUFFER_POOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAB16K</td>
<td>BUFF16K</td>
</tr>
<tr>
<td>TEMPSYS16K</td>
<td>BUFF16K</td>
</tr>
<tr>
<td>TEMPSYS32K</td>
<td>BUFF32K</td>
</tr>
<tr>
<td>TAB8K</td>
<td>BUFF8K</td>
</tr>
<tr>
<td>TEMPSYS8K</td>
<td>BUFF8K</td>
</tr>
<tr>
<td>SYSCATSPACE</td>
<td>IBMDEFAULTBP</td>
</tr>
<tr>
<td>TEMPSPACE1</td>
<td>IBMDEFAULTBP</td>
</tr>
<tr>
<td>USERSPACE1</td>
<td>IBMDEFAULTBP</td>
</tr>
<tr>
<td>SYSTOOLSPACE</td>
<td>IBMDEFAULTBP</td>
</tr>
<tr>
<td>SYSTOOLSTMPSPACE</td>
<td>IBMDEFAULTBP</td>
</tr>
</tbody>
</table>

The IBMDEFAULTBP buffer pool reads pages from each table in every table space. Due to the high number of reads, IBMDEFAULTBP flushes pages out to disk more frequently to make room for pages in other tables, which increases the overall number of input/outputs.

The following list of steps is a high-level view of the procedure to reconfigure the default data model:

1. Run the out-of-the-box (baseline) configuration, which simulates a workload that is accessed by a high number of concurrent users.
2. Analyze the baseline results to determine how to reconfigure the data model.
3. Run the new configuration.
4. Compare the results of the out-of-the-box configuration to the new data model.

### Understanding the workload

Characteristics of the workload include:

- The out-of-the-box configuration of the Aurora start store.
- The Rational Performance Tester that is used to simulate concurrent users in the WebSphere Commerce workload.
- The number of users that browse and place orders on the site, and increase CPU usage to between 60% and 70%.
- The distribution of user groups and functions. Users of the system are divided into two groups: registered users and guest users. The percentage of users in each category and their corresponding activities on the site is outlined in the following table:

### Table 1. Workload description

<table>
<thead>
<tr>
<th>User Group</th>
<th>Function</th>
<th>Weighted Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registered Users (80%)</td>
<td>Browse</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>Add to Cart</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>Order</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td>Simple Search</td>
<td>10%</td>
</tr>
<tr>
<td>Guest Users (20%)</td>
<td>Browse</td>
<td>20%</td>
</tr>
</tbody>
</table>
For each user group, placing an order appropriates 50% of the workload. Placing an order is an insert intensive workload that stresses the database and increases CPU usage on the database server. In an order focused workload, page view throughput is further affected by improved query response time. However, in a browsing focused workload, where caching plays a large role, the impact on page view throughput is diminished.

- The "think time" (or the pause between one HTTP request and the next), is 1 second, which significantly stresses the application.

### Understanding the environment

The environment contains two AIX POWER7 machines, each with four 3612 MHz cores and 5 GB of physical memory. The first machine contains WebSphere Commerce, Feature Pack 7, and a Solr search engine. The second contains DB2, V9.7.0.7, and the WebSphere Commerce database. There are three disks on the database side: disks 0, 1 and 2. The database is created such that all the table spaces are distributed across disks 1 and 2.

### Understanding the basic principals and conditions

Before delving into the analysis of the workload and the data model, there are certain basic principals and conditions that must be understood:

- Small tables that are highly read, such as configuration tables, must be separated. When small tables are placed in their own buffer pool, their pages are loaded in memory and can remain there for a long time without competition from other pages.
- Separating indexes and data into separate table spaces and buffer pools helps to avoid situations in which index pages are forced out from the buffer pool to make space for a table read operation.
- High volume tables with high insert, high update, or high delete activity must also be separated. If these tables are not separated, pages in the buffer pool from other small, highly read tables are flushed out from the buffer pool and read back in repeatedly.
- Tables that are accessed simultaneously are candidates for grouping.

Not all of these principles can be applied in every environment. In an environment with less physical memory, you might avoid separating indexes, data, and large objects for all tables. You might also avoid separating large highly active tables (tables with high insert, high update, or high delete activity).

### Changing the data model

Using this actual environment as an example, a set of queries are run on the WebSphere Commerce database after the baseline run is completed. These queries show:
• Small tables that are highly read.
• Tables with high read, update, insert, and delete activity.
• Whether to upgrade a table to a higher page size.

The queries use the DB2 table function, mon_get_table, to gather monitoring metrics. For more information about this function, see MON_GET_TABLE table function - get table metrics in the DB2 for Linux UNIX and Windows 9.7.0 IBM Knowledge Center.

Step 1: Query the number of rows that are read versus cardinality

The first query retrieves the number of rows read, the number of rows in the table, and the number of reads for each row. This query shows small and medium tables with high read activities. A line is fetched by this query if the number of reads is greater than 10,000 times the cardinality of the table.

Listing 2. Example query for number of rows that are read versus cardinality

```
select substr(t.tabname,1,30) as tabname,
       rows_read,
       card,
       case
           when card > 0 then rows_read/card else NULL
       end as rr_per_card
from table (mon_get_table(NULL, NULL, -2)) as mgt,
     syscat.tables t
where t.tabschema=mgt.tabschema and
     t.tabname=mgt.tabname and
     rows_read > 0 and
     rows_read > 10000*card and
     card > 0 and
     volatile !='C' and
     t.tabschema='WCS'
order by 4 desc
with ur
```

Listing 3. Example output for number of rows that are read versus cardinality

```
<table>
<thead>
<tr>
<th>TABNAME</th>
<th>ROWS_READ</th>
<th>CARD</th>
<th>RR_PER_CARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTRACT</td>
<td>129742408</td>
<td>1198</td>
<td>108299</td>
</tr>
<tr>
<td>TRANSPORT</td>
<td>730855</td>
<td>14</td>
<td>52283</td>
</tr>
<tr>
<td>TERMCOND</td>
<td>8795649</td>
<td>217</td>
<td>40532</td>
</tr>
<tr>
<td>CATENCALCD</td>
<td>414297</td>
<td>13</td>
<td>31869</td>
</tr>
<tr>
<td>CSEDITATT</td>
<td>365491</td>
<td>14</td>
<td>26106</td>
</tr>
<tr>
<td>SEOPAGEDEFOVR</td>
<td>79046</td>
<td>4</td>
<td>19761</td>
</tr>
<tr>
<td>ATTR</td>
<td>10883049</td>
<td>646</td>
<td>16879</td>
</tr>
<tr>
<td>ATTRVAL</td>
<td>91414961</td>
<td>7590</td>
<td>12044</td>
</tr>
<tr>
<td>FACET</td>
<td>118310</td>
<td>10</td>
<td>11831</td>
</tr>
</tbody>
</table>
```

The table, CONTRACT is at the top of the list. It has 1,198 rows, where each row is read 108,299 times. If CONTRACT is put in a separate table space with a dedicated buffer pool, each time a row is read in memory, the probability for that row to remain in memory for a prolonged period is increased. There is almost no competition from other rows in the other tables in this buffer pool. Thus, the data is not flushed to disk to free space in the buffer pool. After a few reads from CONTRACT, its rows are loaded permanently in memory, which reduces the number of input/outputs, and increases the speed of table reads.
Separating small tables with multiple reads into their own buffer pool

When small tables with multiple reads are placed in the same buffer pool as large tables, their pages are flushed away from the buffer pool more frequently to make room for pages that are read from large tables. The pages from the small tables are then loaded from disk repeatedly, which then decreases performance.

TRANSPORT, CATENCALCD, CSEDITATT, SEOPAGEDEFOVR, and FACET are small tables that are read multiple times throughout the run. If you place these tables together in the same table space with a dedicated buffer pool, the queries' ability to fetch their data is boosted.

The tables, TERMCOND and CONTRACT are placed together because both are accessed when a connected user retrieves the contracts, terms, and conditions that are associated with that user. Tables ATTR and ATTRVAL are also placed together for the same reason.

Based on the query result, the new arrangement follows:

Table 2. New tables and table spaces arrangement

<table>
<thead>
<tr>
<th>Table Space</th>
<th>Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBSP_CONT</td>
<td>CONTRACT and TERMCOND</td>
</tr>
<tr>
<td>TBSP_SMALLTAB</td>
<td>TRANSPORT, CATENCALCD, CSEDITATT, SEOPAGEDEFOVR, and FACET</td>
</tr>
<tr>
<td>TBSP_ATTR</td>
<td>ATTR and ATTRVAL</td>
</tr>
</tbody>
</table>

Step 2: Query for the most highly read tables

The following query provides the top 20 highly read tables. The query flags tables where the number of rows that are read is higher than the average number of rows that are read plus two times the standard deviation.

Listing 4. Example query for the most highly read tables

```sql
with read_stats as
(
    select decimal(avg(rows_read) - 2 * stddev(rows_read),20,2) as avgLess2stddev,
            decimal(avg(rows_read) + 2 * stddev(rows_read),20,2) as avgPlus2stddev
    from table (mon_get_table(NULL,NULL,-1))
    where tabschema='WCS'
)
select substr(tabname,1,30) as tabname,
       rows_read,
       case
           when rows_read > read_stats.avgPlus2stddev then 'High'
           when rows_read < read_stats.avgLess2stddev then 'Low'
           else NULL
       end as flag
from table (mon_get_table(NULL,NULL,-1)),
       read_stats
where tabschema='WCS'
order by rows_read desc
fetch first 20 rows only
with ur
```
Listing 5. Example output for the most highly read tables

<table>
<thead>
<tr>
<th>TABNAME</th>
<th>ROWS_READ</th>
<th>FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTRACT</td>
<td>129742408</td>
<td>High</td>
</tr>
<tr>
<td>ATTRVAL</td>
<td>91414961</td>
<td>High</td>
</tr>
<tr>
<td>ATTRVALDESC</td>
<td>50345662</td>
<td>High</td>
</tr>
<tr>
<td>CATENTRY</td>
<td>18929887</td>
<td>High</td>
</tr>
<tr>
<td>PRSETCEREL</td>
<td>13280000</td>
<td>-</td>
</tr>
<tr>
<td>CATENTRYATTR</td>
<td>12647082</td>
<td>-</td>
</tr>
<tr>
<td>ATTRDESC</td>
<td>10820066</td>
<td>-</td>
</tr>
<tr>
<td>ATTR</td>
<td>10883649</td>
<td>-</td>
</tr>
<tr>
<td>RECEIPT</td>
<td>9045762</td>
<td>-</td>
</tr>
<tr>
<td>CATENTDESC</td>
<td>8826013</td>
<td>-</td>
</tr>
<tr>
<td>TERMCOND</td>
<td>8795649</td>
<td>-</td>
</tr>
<tr>
<td>ITEMSPC</td>
<td>8114183</td>
<td>-</td>
</tr>
<tr>
<td>OFFER</td>
<td>7804132</td>
<td>-</td>
</tr>
</tbody>
</table>

Tables CONTRACT and ATTRVAL are at the top of the list. They are already flagged to be moved to different table spaces.

Table ATTRVALDESC is also highly active. It must join ATTR and ATTRVAL in table space, TBSP_ATTR, because it is also being accessed with ATTR and ATTRVAL simultaneously by users on the system.

Because the table, CATENTRY, is flagged by the query makes it a candidate to be separated. A further assessment is needed to see whether this table must be in one of the new table spaces that was already created in Step 1, or in its own table space.

CATENTRY tends to become a large table on e-commerce sites because it contains all catalog entries, such as products, items, packages, and bundles. Placing this table in an existing table space that contains small and medium tables with low cardinality (TBSP_CONT, TBSP_SMALLTAB, and TBSP_ATTR), can cause the pages from these tables to be forced off from the buffer pool and then be loaded back in more frequently, which decreases performance. As a result, it is necessary for CATENTRY to be separated into its own table space.

In Step 2, one table is added to the table space, TBSP_ATTR, and CATENTRY is added to a new table space, TBSP_CAT.

Based on the query result, the new arrangement follows:

**Table 3. New tables and tables spaces arrangement**

<table>
<thead>
<tr>
<th>Tables</th>
<th>Table Spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTRACT and TERMCOND</td>
<td>TBSP_CONT</td>
</tr>
<tr>
<td>TRANSPORT, CATENCALCD, CSEDITATT, SEOPAGEDEFOVR, and FACET</td>
<td>TBSP_SMALLTAB</td>
</tr>
<tr>
<td>ATTR, ATTRVAL, and ATTRVALDESC</td>
<td>TBSP_ATTR</td>
</tr>
<tr>
<td>CATENTRY</td>
<td>TBSP_CAT</td>
</tr>
</tbody>
</table>
Step 3: Query for the tables with the highest update activity

This query provides the top 20 tables with highest update activity. This query flags the tables where the number of rows that are updated is higher than the average number of rows that are updated plus two times the standard deviation.

Listing 6. Example query for tables with the highest update activity

```sql
WITH read_stats AS
  (SELECT decimal(avg(rows_updated) - 2*stddev(rows_updated),20,2) AS avgLess2stddev,
   decimal(avg(rows_updated) + 2*stddev(rows_updated),20,2) AS avgPlus2stddev
   FROM table (mon_get_table(NULL,NULL,-1))
   WHERE tabschema='WCS'
  )
SELECT substr(tabname,1,30) AS tabname,
   rows_updated,
   CASE
      WHEN rows_updated > read_stats.avgPlus2stddev THEN 'High'
      WHEN rows_updated < read_stats.avgLess2stddev THEN 'Low'
      ELSE NULL
   END AS flag
FROM table (mon_get_table(NULL,NULL,-1)),
   read_stats
WHERE tabschema='WCS' AND
   rows_updated > 0
ORDER BY rows_updated DESC
FETCH FIRST 20 ROWS ONLY
WITH UR
```

Listing 7. Example output for tables with the highest update activity

<table>
<thead>
<tr>
<th>TABNAME</th>
<th>ROWS_READ</th>
<th>FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDADJDSC</td>
<td>438204</td>
<td>High</td>
</tr>
<tr>
<td>ORDERITEMS</td>
<td>190067</td>
<td>High</td>
</tr>
<tr>
<td>ORDIADJUST</td>
<td>121449</td>
<td>High</td>
</tr>
<tr>
<td>ORDERS</td>
<td>55710</td>
<td>High</td>
</tr>
<tr>
<td>PPCEXTDATA</td>
<td>39648</td>
<td></td>
</tr>
<tr>
<td>ORDADJUST</td>
<td>33708</td>
<td></td>
</tr>
<tr>
<td>SUBORDERS</td>
<td>21996</td>
<td></td>
</tr>
</tbody>
</table>

The tables ORDADJDSC, ORDERITEMS, ORDIADJUST, and ORDERS are flagged as having the highest update activity. However, must these tables be separated? Because these tables contain details about orders and because the workload is order intensive, the tables grow quickly throughout the run. If they are placed into one separate table space, the buffer pool that is linked to this table space is adjusted many times by the DB2 Self Tuning Memory Manager (STMM).

When to avoid separating active tables into new table spaces

In an environment with little memory, creating more buffer pools with highly active tables causes an increase in CPU processing on the database server. As a result, it isn’t necessary to separate active tables into new table spaces in this sort of environment. Throughout the run, the buffer pools are shrunk and enlarged repeatedly and more frequently while STMM is on to make space for either the new buffer pools or the old ones.

If there is no physical memory available to increase the size of the buffer pool by STMM, some pages are forced back to disk. This situation is likely to happen in an actual environment, which
causes an increase in CPU processing and input/outputs. Therefore, these tables are not moved in this exercise. If the environment for this test had more physical memory, the tables would need to be separated.

**Step 4: Query for tables with the highest insert activity**

This query provides the top 20 tables with the highest insert activity. Like the queries above, it flags the tables where the number of rows that are updated is higher than the average number of rows that are updated plus two times the standard deviation.

**Listing 8. Example query for tables with the highest insert activity**

```sql
with read_stats as
(
  select decimal(avg(rows_inserted)-2*stddev(rows_inserted),20,2) as avgLess2stddev,
         decimal(avg(rows_inserted)+2*stddev(rows_inserted),20,2) as avgPlus2stddev
  from table (mon_get_table(NULL,NULL,-1))
  where tabschema='WCS'
)
select substr(tabname,1,30) as tabname,
       rows_inserted,
       case
         when rows_inserted > read_stats.avgPlus2stddev then 'High'
         when rows_inserted < read_stats.avgLess2stddev then 'Low'
         else NULL
       end as flag
from table (mon_get_table(NULL,NULL,-1))
       read_stats
where tabschema='WCS' and
     rows_inserted > 0
order by rows_inserted desc
fetch first 20 rows only
with ur
```

**Listing 9. Example output for tables with the highest update activity**

<table>
<thead>
<tr>
<th>TABNAME</th>
<th>ROWS_INSERTED</th>
<th>FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTXDATA</td>
<td>48876</td>
<td>-</td>
</tr>
<tr>
<td>PPCSEXDATA</td>
<td>39648</td>
<td>-</td>
</tr>
<tr>
<td>ORDPICKHST</td>
<td>25922</td>
<td>-</td>
</tr>
<tr>
<td>ORDADJDSC</td>
<td>24570</td>
<td>-</td>
</tr>
<tr>
<td>BUSEVENT</td>
<td>9711</td>
<td>-</td>
</tr>
<tr>
<td>CTXMGMT</td>
<td>6198</td>
<td>-</td>
</tr>
<tr>
<td>ORDERITEMS</td>
<td>4568</td>
<td>-</td>
</tr>
<tr>
<td>ORDIADJUST</td>
<td>4568</td>
<td>-</td>
</tr>
</tbody>
</table>

No table is marked as having high insert activity, so no action is taken.

**Step 5: Query for tables with the highest delete activity**

This query provides the top 20 tables with the highest delete activity. Like the queries above, it flags the tables where the number of rows that are updated is higher than the average number of rows that are updated plus two times the standard deviation.

**Listing 10. Example query for tables with the highest delete activity**

```sql
with read_stats as
(
  select decimal(avg(rows_deleted)-2*stddev(rows_deleted),20,2) as avgLess2stddev,
         decimal(avg(rows_deleted)+2*stddev(rows_deleted),20,2) as avgPlus2stddev
)
select substr(tabname,1,30) as tabname,
       rows_deleted,
       case
         when rows_deleted > read_stats.avgPlus2stddev then 'High'
         when rows_deleted < read_stats.avgLess2stddev then 'Low'
         else NULL
       end as flag
from table (mon_get_table(NULL,NULL,-1))
       read_stats
where tabschema='WCS' and
     rows_deleted > 0
order by rows_deleted desc
fetch first 20 rows only
with ur
```
Listing 11. Example output for tables with the highest delete activity

<table>
<thead>
<tr>
<th>TABNAME</th>
<th>ROWS_DELETED</th>
<th>FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDPICKHST</td>
<td>21830</td>
<td>High</td>
</tr>
<tr>
<td>BUSEVENT</td>
<td>9721</td>
<td>High</td>
</tr>
<tr>
<td>CTXDATA</td>
<td>384</td>
<td>-</td>
</tr>
<tr>
<td>SCHACTIVE</td>
<td>99</td>
<td>-</td>
</tr>
<tr>
<td>CTXMGMT</td>
<td>46</td>
<td>-</td>
</tr>
</tbody>
</table>

Tables ORDPICKHST and BUSEVENT are at the top of the list. ORDPICKHST has the same characteristics as the tables mentioned in Step 3. The same justifications that were given in Step 3 also apply here. As a result, ORDPICKHST is not moved.

BUSEVENT is a table that is used to store business event objects. BUSEVENT can be volatile throughout the run. If separated into a dedicated table space it can cause an increase in CPU usage and extra input/outputs. Therefore, BUSEVENT is not moved to separate table space.

Step 6: Change the page size of some tables

Table space TBSP_CONT contains the tables, CONTRACT, and TERMCOND. CONTRACT is in the TAB8K table space, which has a page size of 8 K, and TERMCOND is in USERSPACE1, which has a page size of 4 K. The remainder of the tables is in USERSPACE1, which is a 4 K table space. The new table space size for each table must be 4 K.

The following query gives more details about the page sizes of the tables that were picked to be moved to different table spaces in previous steps. The query uses the GetRowSize routine, which calculates the row size of a table. For more information about this method, see "Size matters: A handy routine to compute the defined row size of a table", a blog entry by Serge Rielau and Rick Swagerman.

Listing 12. Example query to determine whether a table requires page size adjustment

```sql
select substr(tab.tabname,1,20) as tabname,
       tab.card,
       GetRowSize(tab.tabschema, tab.tabname) as rowsize,
       case
            when tbsp.pagesize = 4096 then '4K'
       end as flag
from table (mon_get_table(NULL,NULL,-1))
    where tabschema='WCS'
)
```
when tbsp.pagesize = 8192 then '  8K'
when tbsp.pagesize = 16384 then ' 16K'
when tbsp.pagesize = 32768 then ' 32K'
end as pagesize,
case
    when tab.avgrowsize = 0 then null
  else tbsp.pagesize/tab.avgrowsize
end as avg_rows_per_page,
case
    when GetRowSize(tab.tabschema, tab.tabname) = 0 then null
  else tbsp.pagesize/GetRowSize(tab.tabschema, tab.tabname)
end as min_rows_per_page
from syscat.tables tab,
syscat.table spaces tbsp
where tab.tabname in ('CONTRACT', 'TERMCOND', 'ATTR', 'ATTRVAL', 'ATTRVALDESC',
                     'TRANSPORT', 'CATENCALCD', 'CSEDITATT', 'SEOPAGEDEFOVR',
                     'FACET', 'CATENTRY') and
  tab.type = 'T' and
  tab.tabschema = 'WCS' and
  tab.tbspaceid = tbsp.tbspaceid
order by 5, 6

Listing 13. Example output of tables that require page size adjustment

<table>
<thead>
<tr>
<th>TABNAME</th>
<th>CARD</th>
<th>ROWSIZE</th>
<th>PAGESIZE</th>
<th>AVG_ROWS_PER_PAGE</th>
<th>MIN_ROWS_PER_PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATENTRY</td>
<td>553321</td>
<td>1441</td>
<td>4K</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>TERMCOND</td>
<td>217</td>
<td>3978</td>
<td>4K</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>ATTRVALDESC</td>
<td>33831</td>
<td>3372</td>
<td>4K</td>
<td>31</td>
<td>1</td>
</tr>
<tr>
<td>ATTR</td>
<td>640</td>
<td>604</td>
<td>4K</td>
<td>34</td>
<td>6</td>
</tr>
<tr>
<td>CSEDITATT</td>
<td>14</td>
<td>546</td>
<td>4K</td>
<td>45</td>
<td>7</td>
</tr>
<tr>
<td>FACET</td>
<td>10</td>
<td>587</td>
<td>4K</td>
<td>48</td>
<td>6</td>
</tr>
<tr>
<td>CONTRACT</td>
<td>1198</td>
<td>4316</td>
<td>8K</td>
<td>54</td>
<td>1</td>
</tr>
<tr>
<td>ATTRVAL</td>
<td>7590</td>
<td>555</td>
<td>4K</td>
<td>57</td>
<td>7</td>
</tr>
<tr>
<td>TRANSPORT</td>
<td>14</td>
<td>537</td>
<td>4K</td>
<td>69</td>
<td>7</td>
</tr>
<tr>
<td>SEOPAGEDEFOVR</td>
<td>4</td>
<td>158</td>
<td>4K</td>
<td>77</td>
<td>25</td>
</tr>
<tr>
<td>CATENCALCD</td>
<td>13</td>
<td>41</td>
<td>4K</td>
<td>80</td>
<td>99</td>
</tr>
</tbody>
</table>

11 record(s) selected.

Although table CATENTRY has 10 VARCHAR columns where three have a size of 254 bytes, they are not considered to be large columns. The page size of CATENTRY is not changed and thus, TBSP_CAT remains a 4 K table space.

Alternatively, TERMCOND, must change its page size from 4 K to 8 K if it is moved to the same table space as CONTRACT because the page size of the latter is 8 K. TERMCOND also has three VARCHAR columns, where one column has a size of 3200, which is considered to be large. When the column is fully populated, a page cannot hold more than one row, which wastes much space. Therefore, TERMCOND must be moved to a table space with a higher page size.

The new arrangement that is based on the query result is as follows:

Table 4. Final tables and table spaces arrangement

<table>
<thead>
<tr>
<th>Tables</th>
<th>Page size</th>
<th>Table spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTRACT and TERMCOND</td>
<td>8K</td>
<td>TBSP_CONT</td>
</tr>
<tr>
<td>TRANSPORT, CATENCALCD, CSEDITATT, SEOPAGEDEFOVR, and FACET</td>
<td>4K</td>
<td>TBSP_SMALLTAB</td>
</tr>
</tbody>
</table>
Results and comparisons

To understand how system performance improved as a result of changing the data model, compare the system performance of a run before and after reconfiguring the data model. Database and web metrics are measured and analyzed.

1. **Database metrics** are collected from the `nmon` tool. Smaller values in each metric are optimal for your system.
   - **Input-output (I/O)** is the number of disk reads (kilobytes read from disk per second), the number of disk writes (kilobytes written to disk per second), and the number of input/output operations that are performed per second (IOPS).
   - **CPU usage** includes the CPU Wait percent, CPU Idle percent, and overall percentage of CPU usage by the system and by users.

2. **Web metrics** are collected from Rational Performance Tester and WebSphere administrative (`wsadmin`) scripting program.

   Greater values in the following four metrics are optimal for your system.
   - **Throughput** is the amount of data that is transferred between WebSphere Commerce servers and the clients (users who are connected to a WebSphere Commerce site). Throughput is collected from the Rational Performance Tester report.
   - **Page views per second** is the average number of pages of WebSphere Commerce site that is viewed per second. Page views per second is collected from WebSphere administrative scripting program.
   - **Transactions per second** is the average number of transactions per second. A transaction is any complete operation. For example, a user completes one transaction if that user logged in, navigated to a top category of items, navigated to a secondary category, viewed a product, and then added that product to the shopping cart.
   - **Total transactions that are performed** is the total number of transactions and the number of transactions per second are collected by the Rational Performance Tester report.

Smaller values in the following four metrics are optimal for your system.
   - **Average number of connections** is collected from the WebSphere connection pool throughout each run. Using fewer connections means saving the cost of allocating and releasing the connections.
   - **Average number of web containers**: WebSphere Commerce allocates threads from Web Container pool to handle incoming HTTP requests. Using less Web Containers means saving the cost of allocating and destroying the threads and keeping the threads for future large requests.
   - **Average query time, or Java Database Connectivity (JDBC) time** is the average time for all the queries that are executed against the database.
• **Web requests response time** is the average response time for each action, like the response time of login to WebSphere Commerce storefront, browse to a top category, browse to a lower category, display a product, add an item to a cart.

**Database Metrics**

**Table 5. Input/Output in the database server**

<table>
<thead>
<tr>
<th></th>
<th>Disk Read (kbps)</th>
<th>Disk Write (kbps)</th>
<th>I/Os (per second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out-of-the-Box (OOTB)</td>
<td>148.3 kbps</td>
<td>362.4 kbps</td>
<td>77.8 kbps</td>
</tr>
<tr>
<td>Results after moving tables (TBSP)</td>
<td>246.3 kbps</td>
<td>463.6 kbps</td>
<td>103.0 kbps</td>
</tr>
<tr>
<td>TBSP vs OOTB (%)</td>
<td>66.08%</td>
<td>27.92%</td>
<td>32.39%</td>
</tr>
</tbody>
</table>

**Table 6. CPU usage in the database server**

<table>
<thead>
<tr>
<th></th>
<th>User (%)</th>
<th>Sys (%)</th>
<th>Wait (%)</th>
<th>Idle (%)</th>
<th>CPU (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out-of-the-Box (OOTB)</td>
<td>44.32%</td>
<td>21.52%</td>
<td>0.21%</td>
<td>33.96%</td>
<td>65.84%</td>
</tr>
<tr>
<td>Results after moving tables (TBSP)</td>
<td>41.16%</td>
<td>20.85%</td>
<td>0.14%</td>
<td>37.84%</td>
<td>62.01%</td>
</tr>
<tr>
<td>TBSP vs OOTB (%)</td>
<td>7.68%</td>
<td>3.21%</td>
<td>50.00%</td>
<td>-10.25%</td>
<td>6.18%</td>
</tr>
</tbody>
</table>

In the database server, the number of input/outputs increased by 32.39%. However, the CPU usage decreased by 6.18%.

**Web metrics**

**Table 7. Web metrics comparisons**

<table>
<thead>
<tr>
<th></th>
<th>Page view (per sec)</th>
<th>Avg # of Connections</th>
<th>Avg # of web Containers</th>
<th>Avg JDBC Time</th>
<th>Transactions (per second)</th>
<th>Transactions (all)</th>
<th>Total Page Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out-of-the-box (OOTB)</td>
<td>20.70</td>
<td>14.5</td>
<td>21.12</td>
<td>1.64</td>
<td>1.36</td>
<td>5,007</td>
<td>76,290</td>
</tr>
<tr>
<td>Results after moving tables (TBSP)</td>
<td>29.70</td>
<td>10.17</td>
<td>17.25</td>
<td>0.68</td>
<td>1.96</td>
<td>7,222</td>
<td>109,872</td>
</tr>
<tr>
<td>TBSP vs OOTB (%)</td>
<td>43.47%</td>
<td>-29.8%</td>
<td>-18.3%</td>
<td>-58.5%</td>
<td>44.11%</td>
<td>44.23%</td>
<td>44.02%</td>
</tr>
</tbody>
</table>

These results display a clear increase in performance. The average query time decreased by 58.50%. The number of transactions increased by 44.23%. The number of connections and the number of web containers decreased 29.8% and 18.3%, respectively. The database server can now do more in less time.

**Response time comparison**

The following chart depicts the top 10 response times. The lower the figure, the more favorable the result. Activities marked "Run #1" represent the response times of the out-of-the-box configuration. Activities marked "Run #2" represent the response times after the tables are moved to their new configuration.
Figure 1. Response Time

Throughput

Table 8. Web metrics comparisons - Part 1

<table>
<thead>
<tr>
<th></th>
<th>Throughput (bytes received)</th>
<th>Throughput (bytes sent)</th>
<th>Total Page Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out-of-the-box (OOTB)</td>
<td>7,698,740,698</td>
<td>146,103,813</td>
<td>76,290</td>
</tr>
<tr>
<td>Results after moving tables (TBSP)</td>
<td>11,206,506,958</td>
<td>210,432,229</td>
<td>109,872</td>
</tr>
</tbody>
</table>

The blue line represents the rate of throughput of the out-of-the-box configuration. The red line represents the rate of throughput after the tables are moved to their new configuration.
Figure 2. Throughput

The blue line represents the transaction completion rate of the out-of-the-box configuration. The red line represents the transaction completion rate after the tables are moved to their new configuration.

Transactions

The blue line represents the transaction completion rate of the out-of-the-box configuration. The red line represents the transaction completion rate after the tables are moved to their new configuration.
Overall performance increased after the reconfiguration. While there is a slight increase in the number of input/outputs per second in the database server, there is a decrease in CPU usage and increase in throughputs. This increase in input/outputs is expected because the database server can now do more in a shorter amount of time.

**Generating a script to change the data model**

A tool is provided with the article to generate a script to move the tables to different table spaces. The tool is a Perl script named `generate_move_tables_script.pl`. Tool parses a text properties file and generates a shell script to create the new table spaces and buffer pools and generate table move commands.

To use the tool and change the default data model, complete the following procedure:

1. In "Changing the data model", you ran queries to determine what reconfiguration of the table spaces and tables is optimal for your system. Those queries result in the suggested new arrangement. Table spaces are displayed with the suggested page size and configuration of tables:
TBSP_CONT (8 K) => CONTRACT, TERMCOND
TBSP_SMALLTAB (4 K) => TRANSPORT, CATENCALCD, CSEDITATT, SEOPAGEDEF0VR, FACET
TBSP_ATTR (4 K) => ATTR, ATTRVAL, and ATTRVALDESC
TBSP_CAT (4 K) => CATENTRY

2. Create a properties file with the text editor of your choice. For each new table space configuration outlined in Step 1, enter the new data model in the following format:

```
T,(table names separated by commas)
D,(pagesize),(table space name),(container for data),(container for indexes),(container for large objects)
I,N
L,N
```

Where,

- "T" defines the tables to be moved to the new table space.
- "D" defines the data table spaces
- "I" defines the index table spaces. To indicate that an index table space will not be created, enter "N".
- "L" defines the large table spaces. To indicate that a large table space will not be created, enter "N".

If you are using automatic table spaces, do not enter anything in the containers for data, indexes, or large tables. For example, for the table space, TBSP_CONT, the line `TBSP_CONT (8 K) => CONTRACT, TERMCOND` becomes:

```
T,CONTRACT,TERMCOND
D,8,TBSP_CONT,,,
I,N
L,N
```

3. Complete entering the new data model. Based on the analysis run throughout the article, the properties file should resemble the following output:

```
T,CONTRACT,TERMCOND
D,8,TBSP_CONT,,,
I,N
L,N

T,ATTR,ATTRVAL,ATTRVALDESC
D,4,TBSP_ATTR,,,
I,N
L,N

T,FACET,CSEDITATT,CATENCALCD,SEOPAGEDEF0VR,TRANSPORT
D,4,TBSP_SMALLTAB,,,
I,N
L,N

T,CATENTRY
D,4,TBSP_CAT,,,
I,N
L,N
```

Save the properties file as `properties.txt` in the same directory as the `generate_move_tables_script.pl` file.

4. To run the tool, enter the following command into the command-line interface, specifying the database name and schema:

`generate_move_tables_script.pl dbname schema`
The tool uses the original page size and settings that are defined in `EXTENSIZE`, `OVERHEAD`, and `TRANSFERRATE`, for each table space. In the specified database and schema, the tool creates a DMS (database managed space) table space with one container and a page size of 5000. After you run the tool, a shell script file, `move_tables.sh`, is generated in the same folder.

5. Run the newly generated shell script, `move_tables.sh` to reconfigure the data model. If you run the tool from the command-line interface, the new table spaces and buffer pools that were defined in the properties file are created, and the tables are moved to their new spaces. The script also includes queries to gather data before and after the reconfiguration for statistical purposes.

**Conclusion**

Using an actual environment as an example, the workload is analyzed to determine how best to change the data model to improve performance in a WebSphere Commerce environment. You learned how to analyze the data model to configure it for better database performance. You also learned why certain tables had to be moved to new table spaces. Finally, you compared analysis results that were obtained from the out-of-the-box configuration to results obtained from the data model reconfiguration, proving an improvement in performance. In addition, you were provided with a tool that generates commands to reconfigure the data model. This entire process can also be applied to other data models to improve performance in different environments.

**Acknowledgements**

The author thanks and acknowledges the following people from the WebSphere Commerce development team and from the Commerce Performance team for their help, reviews, input, and valuable advice.

- Fahad Javed - WebSphere Commerce Development, Database focal point
- Kevin Yu - Executive ITS, Smarter Commerce Performance Architect
- Sun Lei - Senior Software Engineer, Performance Analyst
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data model reconfiguration script</td>
<td>Tool_to_pick_and_move_the_tables.zip</td>
<td>9KB</td>
</tr>
</tbody>
</table>
Resources

Learn

- Read more about queries to analyze the workload and data model in "Redesigning Table spaces in an Existing Database" (DB2 Commerce 2014)
- "DB2 Basics: Table spaces and buffer pools" (developerWorks 2010)
- Read more about "WebSphere Commerce 7.0" in the WebSphere Commerce Knowledge Center documentation.
- Read more about the "ADMIN_MOVE_TABLE procedure - Move tables online" in the DB2 Knowledge Center documentation.
- Read more about the "CREATE BUFFERPOOL statement" in the DB2 Knowledge Center documentation.
- Read more about the "CREATE TABLE SPACE statement" in the DB2 Knowledge Center documentation.
- Read more about the "MON_GET_TABLE table function - get table metrics" in the DB2 Knowledge Center documentation.
- Read more about "WebSphere Commerce 7.0" in the WebSphere Commerce Knowledge Center documentation.
- Read more about "DB2 9.7" in the DB2 Knowledge Center documentation
- Read more about "DB2 for Linux, UNIX, and Windows Best Practices".
- Stay current with developerWorks technical events and webcasts focused on various IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools as well as IT industry trends.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.

Get products and technologies

- Download WebSphere Commerce.
- Download DB2 Express-C.

Discuss

- Get involved in the WebSphere Commerce community
About the author

Noureddine Brahimi

Noureddine Brahimi is a Software Performance Analyst with the WebSphere Commerce development team. Noureddine has 17 years of experience in IT, specializing in development, databases, ERPs, and performance analysis. Since joining the WebSphere Commerce team in March 2011, Noureddine worked extensively on ensuring better performance for WebSphere Commerce.