War of the Indices
SQL Server vs. Oracle

Kellyn Pot’Vin-Gorman | Technical Intelligence Manager | January, 2018
Kellyn Pot‘Vin-Gorman
Technical Intelligence Manager, Delphix

• Multi-platform DBA, (Oracle, MSSQL, MySQL, Sybase, PostgreSQL, Informix…)
• Oracle ACE Director, (Alumni)
• Oak Table Network Member
• Idera ACE 2018
• APEX Women in Technology Award, CTA
• STEM education with Raspberry Pi and Python, including DevOxx4Kids, Oracle Education Foundation and Girl Geek Dinners
• President, Rocky Mtn Oracle User Group
• President, Denver SQL Server User Group
• Technical author, instructor and presenter.
• Author, blogger, (http://dbakevlar.com)
1. Background
2. Platform Translations
3. Test Cases and Goal
4. Code and Discussions
5. Analysis and Summary
The What- Version 1

Azure SQL Database vs. AWS with Oracle 11.2.0.4
The What- Version 2
The Why

- It's interesting...to me, at least.
- VM was an issue, so moved everything to the cloud and also worked with Docker.
- Two major players in the database landscape.
- Good apples to “apfel” comparison.
Standard Indexing

• An Index, like an index in a book references data.

SQL Server Indexes
• Clustered Index contains data, one per table
• Non-clustered is just pointers and a table can have up to 999, (version dependent)
• No Clustered index results in table referred to as a heap table.
• Clustered index is sorted upon original build

Oracle
• All indexes are by default heap
• Index Organized Table, (IOT) is closest to a SQL Server Index
• Multiple complex objects to compare to SQL Server
• ROWID is a game changer
Oracle’s ROWID

- BASE64 Format Identifier (A-Za-z0-9+/)
- Oracle has little knowledge of clustered indexes and more so, little need of them.
- ROWIDs are the fastest way of accessing rows in Oracle.

What Information Makes up a ROWID?

- Schema
- Table
- Tablespace
- Row
- Block
Oracle Vs. SQL Server

Similar at High Level

Index = Clustered Index
# Translations Graph

<table>
<thead>
<tr>
<th>ORACLE</th>
<th>SQL SERVER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Organized Table, (IOT)</td>
<td>Clustered Index</td>
<td>physical index storing data in their key values. In SQL Server, there can be only one Clustered index per table.</td>
</tr>
<tr>
<td>Pctfree of block</td>
<td>FillFactor of page</td>
<td>Percent of storage that is allowed filled. There are different times when this is used for each platform.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Query Insert</td>
<td>Ability to populate data with a sequential number</td>
</tr>
<tr>
<td>dbms_random.string block</td>
<td>Replicate</td>
<td>Ability to populate data with string values</td>
</tr>
</tbody>
</table>
**Fill Factor in SQL Server**

**FILLFACTOR** is a setting for indexes in SQL Server. When you create or rebuild an index, you can tell SQL Server what percentage of each 8K data page used in the “leaf” level of the index it should fill up. Default is 80 percent, but for non-OLTP, often recommended to be 100.
PCT Free in Oracle

**PCTFREE** is a block storage parameter used to specify how much space should be left in a database block for future updates. For example, for PCTFREE=10, Oracle will keep on adding new rows to a block until it is 90% full. This leaves 10% for future updates (row expansion). This defaults to 10% for indexes, but can be adjusted via a index rebuild.
SQL Server Index Writes - Not Subject to Page Splits

Newly built index leaf after a single page split

Red arrow is the allocation order
Black arrows are following the logical order
SQL Server Index Processing Impacted by Random Fragmentation

Index leaf level after random inserts/deletes

Red arrow is the allocation order
Black arrows are following the logical order

Fragmentation
In What Case Would You Rebuild?
• Significant # of logically deleted index nodes.
• Inefficient number of gets per access

So yes, there are times and situations you need to rebuild.

The goal here is to see where each platform performs better than the other.
1st Test Case

- Create a table with three columns and two indexes.
- Insert, update and delete different amounts of data.
- Check the storage of our index and any fragmentation, storage anomalies.
- Repeat
- Check the storage repeatedly to see how it has changed—page splits in SQL Server, leaf block splits in Oracle
Goal

1. Inspect the differences and similarities of indexing in both platforms
2. The pros and cons of how index data is stored and used in the database platforms.

http://dbakevlar.com/2017/04/oracle-sql-server-index-comparison/
First Test, Build and Check

The Basics
Oracle Objects

CREATE TABLE ORA_INDEX_TST
(
C1 NUMBER NOT NULL,
C2 VARCHAR2(255),
CREATEDATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP)

;CREATE INDEX PK_INDEXPS ON ORA_INDEX_TST (C1);
CREATE INDEX IDX_INDEXPS ON ORA_INDEX_TST (C2);
ALTER TABLE ORA_INDEX_TST ADD CONSTRAINT OIT_PK
PRIMARY KEY(C1) USING INDEX PK_INDEXPS;

ALTER INDEX IDX_INDEXPS REBUILD PCTFREE 20 INITRANS 5;
ALTER INDEX PK_INDEXPS REBUILD PCTFREE 20 INITRANS 5;
Oracle Support Objects

CREATE SEQUENCE C1_SEQ START WITH 1;

CREATE OR REPLACE TRIGGER C1_BIR
BEFORE INSERT ON ORA_INDEX_TST
FOR EACH ROW
BEGIN
    SELECT C1_SEQ.NEXTVAL
    INTO   :new.C1
    FROM   DUAL;
END;
/

Insert 7 Rows to Fill One Oracle Block

```sql
INSERT INTO ORA_INDEX_TST (C2, CREATEDATE)
VALUES (dbms_random.string('A', 200), SYSDATE);

INSERT INTO ORA_INDEX_TST (C2, CREATEDATE)
VALUES (dbms_random.string('F', 200), SYSDATE);

INSERT INTO ORA_INDEX_TST (C2, CREATEDATE)
VALUES (dbms_random.string('G', 200), SYSDATE);

COMMIT;
```
Check the Block

SQL> ANALYZE INDEX PK_INDEXPS VALIDATE STRUCTURE;

SQL> SELECT LF_BLKS, LF_BLK_LEN, DEL_LF_ROWS, USED_SPACE, PCT_USED FROM INDEX_STATS where NAME='PK_INDEXPS';

<table>
<thead>
<tr>
<th>LF_BLKS</th>
<th>LF_BLK_LEN</th>
<th>DEL_LF_ROWS</th>
<th>USED_SPACE</th>
<th>PCT_USED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7924</td>
<td>0</td>
<td>1491</td>
<td>19</td>
</tr>
</tbody>
</table>

As expected- 1 leaf block for the seven rows.
ALTER INDEX IDX_INDEXPS REBUILD PCTFREE 90 INITRANS 5;

- Yes, we’ve just adjusted the pctfree to 90%!
- Consider what this will do to our index storage now that we’ve rebuilt this allowing for only 10% usage in each block.
- Insert the 8th row:

  INSERT INTO ORA_INDEX_TST (C2, CREATEDATE) VALUES (dbms_random.string('H', 200), SYSDATE);
After Change to Pct Free

SQL> ANALYZE INDEX PK_INDEXPS VALIDATE STRUCTURE;

SQL> SELECT LF_BLKS, LF_BLK_LEN, DEL_LF_ROWS, USED_SPACE, PCT_USED FROM INDEX_STATS where NAME='PK_INDEXPS';

<table>
<thead>
<tr>
<th>LF_BLKS</th>
<th>LF_BLK_LEN</th>
<th>DEL_LF_ROWS</th>
<th>USED_SPACE</th>
<th>PCT_USED</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8229</td>
<td>0</td>
<td>2369</td>
<td>12</td>
</tr>
</tbody>
</table>

Query Time Increase: 2.1 Times Slower
SQL Server’s Turn
CREATE TABLE SQL_INDEX_TST (c1 INT NOT NULL, c2 CHAR (255), createdate DATETIME NOT NULL DEFAULT GETDATE());

CREATE INDEX CL2_INDEX_TST ON SQL_INDEX_TST(C2);
GO

ALTER TABLE SQL_INDEX_TST
ADD CONSTRAINT PK_CLINDX_TST PRIMARY KEY CLUSTERED (c1);
Insert 7 Rows to Fill up Initial SQL Server Page

```sql
INSERT INTO SQL_INDEX_TST(c1,c2) VALUES (1, 'a');
INSERT INTO SQL_INDEX_TST(c1,c2) VALUES (2, 'a');
INSERT INTO SQL_INDEX_TST(c1,c2) VALUES (3, 'a');
INSERT INTO SQL_INDEX_TST(c1,c2) VALUES (4, 'a');
INSERT INTO SQL_INDEX_TST(c1,c2) VALUES (6, 'a');
INSERT INTO SQL_INDEX_TST(c1,c2) VALUES (7, 'a');
GO
```
SELECT

OBJECT_SCHEMA_NAME(ios.object_id) + '.' +
OBJECT_NAME(ios.object_id) as table_name,

i.name as index_name, leaf_allocation_count,
nonleaf_allocation_count

FROM sys.dm_db_index_operational_stats(DB_ID(),
OBJECT_ID('dbo.SQL_INDEX_TST'),NULL, NULL) ios

INNER JOIN sys.indexes i ON i.object_id =
ios.object_id AND i.index_id = ios.index_id;
### Results

<table>
<thead>
<tr>
<th>table_name</th>
<th>index_name</th>
<th>leaf_allocation_count</th>
<th>non_leaf_allocation_count</th>
<th>fill_factor</th>
<th>type_desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo.SQL_INDEX_TST</td>
<td>NULL</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>HEAP</td>
</tr>
<tr>
<td>dbo.SQL_INDEX_TST</td>
<td>CL2_INDEX_TST</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>NONCLUSTERED</td>
</tr>
<tr>
<td>dbo.SQL_INDEX_TST</td>
<td>PK_CLINDEX_TST</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CLUSTERED</td>
</tr>
</tbody>
</table>
ALTER INDEX CL_INDEX_TST ON dbo.SQL_Index_tst
REBUILD WITH (FILLFACTOR = 10);
GO

Now insert the 8th row in:

INSERT INTO SQL_INDEX_TST(c1,c2) VALUES (8, 'a');
GO
Results

Leaf Allocation Count: 8
Fill Factor: 10

Query Time Increase: 2.3 Times
SLOWER
Analysis Says?

- Both use 1 Leaf block with correct fill setting.
- Both show same increases when fill setting at 90%
- Both show similar performance hit due to the change
Second Test, Build and Check

Large Data Loads
Data Loads into Oracle

SQL> Begin
For IDS in 1..1000000
Loop
INSERT INTO ORA_INDEX_TST (C2, CREATEDATE)
VALUES (dbms_random.string('X', 200), SYSDATE);
Commit;
End loop;
End;
/

Check Data and Delete Data

SQL> select count(*) from ora_index_tst;

COUNT(*)
-------
1000008

SQL> delete from ora_index_tst
    where c2 like '%200%';
437 rows deleted.

SQL> commit;
Commit complete.
Check Index

```sql
SELECT LF_BLKKS, LF_BLK_LEN, DEL_LF_ROWS, USED_SPACE, PCT_USED FROM INDEX_S
```

<table>
<thead>
<tr>
<th>LF_BLKKS</th>
<th>LF_BLK_LEN</th>
<th>DEL_LF_ROWS</th>
<th>USED_SPACE</th>
<th>PCT_USED</th>
</tr>
</thead>
<tbody>
<tr>
<td>41227</td>
<td>7924</td>
<td>121</td>
<td>212596009</td>
<td>19</td>
</tr>
</tbody>
</table>
Ending Row Count

SQL> select count(*) from ora_index_tst;

COUNT(*)
---------
995819
Resulting Index Fragmentation and Storage

```sql
SELECT LF_BLKS, LF_BLK_LEN, DEL_LF_ROWS, USED_SPACE, PCT_USED FROM INDEX_STATS;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>LF_BLKS</th>
<th>LF_BLK_LEN</th>
<th>DEL_LF_ROWS</th>
<th>USED_SPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDX_INDEXPS</td>
<td>41342</td>
<td>7996</td>
<td>618</td>
<td>212700845</td>
</tr>
</tbody>
</table>
SQL Server’s Turn
SQL Server Insert

declare @id int
select @id = 9 --already inserted 8 rows
while @id >= 0 and @id <= 1000000
begin
    insert into sql_index_tst (c1,c2) values(@id, 'DKUELKJ' + convert(varchar(7), @id))
    select @id = @id + 1
end

Default Fill Factor- Elapsed Time: 4 minutes, 43 seconds
10% Fill Factor- Elapsed time: 23 minutes, 18 seconds
Check Page Splits in SQL Server

```
SELECT
OBJECT_SCHEMA_NAME(ios.object_id) + '.' +
OBJECT_NAME(ios.object_id) as table_name
,i.name as index_name
,leaf_allocation_count
,nonleaf_allocation_count
FROM sys.dm_db_index_operational_stats(DB_ID(),
OBJECT_ID('dbo.SQL_Index_tst'),NULL, NULL) ios
INNER JOIN sys.indexes i ON i.object_id = ios.object_id
AND i.index_id = ios.index_id;
```
## Page Splits Observed

<table>
<thead>
<tr>
<th>table_name</th>
<th>index_name</th>
<th>leaf_allocation_count</th>
<th>nonleaf_allocation_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo.SQL_INDEX_TST</td>
<td>NULL</td>
<td>6358</td>
<td>0</td>
</tr>
<tr>
<td>dbo.SQL_INDEX_TST</td>
<td>CL2_INDEX_TST</td>
<td>19561</td>
<td>608</td>
</tr>
<tr>
<td>dbo.SQL_INDEX_TST</td>
<td>PK_CLINDX_TST</td>
<td>679</td>
<td>2</td>
</tr>
</tbody>
</table>
SELECT OBJECT_SCHEMA_NAME(ips.object_id) + '.' + OBJECT_NAME(ips.object_id) as table_name,ips.avg_fragmentation_in_percent,ips.fragment_count,page_count FROM sys.dm_db_index_physical_stats(DB_ID(),OBJECT_ID('dbo.SQL_Index_tst'))
### Fragmentation Results

<table>
<thead>
<tr>
<th>table_name</th>
<th>index_id</th>
<th>index_type_desc</th>
<th>avg_fragmentation_in_percent</th>
<th>fragment_count</th>
<th>page_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo.SQL_INDEX_TST</td>
<td>0</td>
<td>HEAP</td>
<td>99.79638709677419</td>
<td>3466</td>
<td>27768</td>
</tr>
<tr>
<td>dbo.SQL_INDEX_TST</td>
<td>2</td>
<td>CLUSTERED INDEX</td>
<td>79.67955863736846</td>
<td>43239</td>
<td>52927</td>
</tr>
<tr>
<td>dbo.SQL_INDEX_TST</td>
<td>3</td>
<td>NONCLUSTERED INDEX</td>
<td>0.4390408645727794</td>
<td>381</td>
<td>2961</td>
</tr>
</tbody>
</table>
Analysis

Inserts and deletes occurred in similar time

SQL Server suffered more fragmentation, but Oracle still experienced fragmentation

Without index maintenance, performance suffered at greater level in SQL Server
Third Test, Build and Check

IOT Vs. Clustered
Clustered Indexes vs. Oracle IOTs

• Oracle Index Organized Tables are most similar to SQL Server Clustered Indexes.

• How do those compare?
Index Organized Table, (IOT)

- Variation of a primary b-tree index
- The index IS the table
- Data is sorted
Oracle IOT Syntax

```
CREATE TABLE ora_tst_iot(
    c1 NUMBER,
    c2 varchar2(255),
    CREATEDATE timestamp DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT pk_ora_iot PRIMARY KEY (c1))
ORGANIZATION INDEX
TABLESPACE users
PCTFREE 20
OVERFLOW TABLESPACE users;
```
Supporting Features

- Created Sequence for PK on C1 column
- Created Trigger to insert next value in C1 on insert.
- The % Threshold set to 20
- No compression
- Loaded from my original table ORA_INDEX_TST

```sql
SQL> insert into ora_tst_iot(c2) select c2 from ora_index_tst;
995830 rows created.
Elapsed: 00:00:04:01
```
Data Load and Validation

SQL> ANALYZE INDEX PK_INDEXPS VALIDATE STRUCTURE;
Index analyzed.

Elapsed: 00:00:00.37

SQL> select index_name from dba_indexes
   2   where table_name='ORA_TST_IOT';

INDEX_NAME
PK_ORA_IOT

SQL> analyze index pk_ora_iot validate structure;
Index analyzed.

SQL> analyze index pk_ora_iot compute statistics;
Index analyzed.

SQL> SELECT LF_BLKS, LF_BLK_LEN, DEL_LF_ROWS,USED_SPACE, PCT_USED FROM

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32115</td>
<td>7996</td>
<td>0</td>
<td>224394262</td>
<td>88</td>
</tr>
</tbody>
</table>
Remove Data-

SQL> select * from ora_tst_iot
   2  where c1=994830;

994830

SBTF02LYEQDFGG2522Q3N3EA2N8IV7SML1MU1IMEG2KLZA6SICGLAVGY2XWADLZSZAHZ0J1
004638IK3JQBW7D92V2ZYQBN49NHJHZR12DM3JWJ1SVWXS76RMBBE90TDUKRZJVLTPIBX5L
VWZTXROKFWYD33R4UID7VXT2NG5ZH5IP9TDQ8G0

10-APR-17 03.09.52.115290 PM

SQL> delete from ora_tst_iot
   2  where c1 >=994820
   3  and c1 <=994830;

11 rows deleted.

SQL> commit;
IOT Vulnerabilities

```
SQL> SELECT 'Chained or Migrated Rows = ' || value
       FROM v$sysstat
       WHERE name = 'table fetch continued row';

Chained or Migrated Rows = 73730
```
Findings

C2 column has significant data and without C1, has a difficult time for access.

ALTER TABLE ORA_IOT_TST REBUILD;

Disable trigger for sequence and then load table with simplified data, then rebuild.

What could possibly go wrong?
After Issuing a Move Statement on an IOT
SQL> select index_name from dba_indexes
    2  where table_name='ORA_TST_IOT';

INDEX_NAME
------------------------
PK_ORA_IOT

SQL> analyze index pk_ora_iot validate structure;
Index analyzed.

SQL> select blocks, height, br_blks, lf_blks from index_stats;

<table>
<thead>
<tr>
<th>BLOCKS</th>
<th>HEIGHT</th>
<th>BR_BLKS</th>
<th>LF_BLKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>32768</td>
<td>3</td>
<td>45</td>
<td>32115</td>
</tr>
</tbody>
</table>
So PCT Free

- As discussed, Pct Free is how much free is to be left at the end of a block...90%...hmm...

```
SQL> alter table ora_tst_iot move pctfree 90;
Table altered.

SQL> analyze index pk_ora_iot validate structure;
Index analyzed.

SQL> SELECT blocks, height, br_blks, lf_blks FROM index_stats;

<table>
<thead>
<tr>
<th>BLOCKS</th>
<th>HEIGHT</th>
<th>BR_BLKS</th>
<th>LF_BLKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>172928</td>
<td>3</td>
<td>228</td>
<td>165630</td>
</tr>
</tbody>
</table>
```

Almost as many leafs as blocks!
Defragment our SQL Server Clustered Index

ALTER INDEX ALL ON SQL_INDEX_TST
REBUILD WITH (ONLINE = ON,
FILLFACTOR = 80, SORT_IN_TEMPDB
= ON, STATISTICS_NORECOMPUTE = ON);

ALTER INDEX ALL ON SQL_INDEX_TST
REORGANIZE;

Increase in performance post reorganize or rebuild:
45% average improvement in tests
Comparisons - Pct Free Vs. Fill Factor

- 10% Fill Factor in SQL Server and 1 million insert: Elapsed time: 23 minutes, 18 seconds
- 90% PCTFree in Oracle and 1 million insert: 7 min, 12 seconds
- 100% Fill Factor in SQL Server and 1 million insert: Elapsed Time: 4 minutes, 43 seconds
- 0% PCTFree in Oracle and 1 million insert: 1 min, 8 seconds
- REBUILD of the Oracle IOT to make it 90% free in each block? Elapsed Time: 8 hrs, 21 minutes, 12 seconds
Analysis

1. After reviewing all the data, only the index without index maintenance showed significant performance degradation in SQL Server vs. Oracle.

2. Introduction of IOT index then created a new scenario where it was obvious there are limitations for Oracle to take advantage of an organized index that stores the data.
Column Store Indexes

Really?
Columnstore Indexes
SQL Server Columnstore Limitations, (2016/2017)

- Can’t have more than 1024 columns, (why would you?? 😉)
- Constraints can’t be part of the columnstore
- Can’t be altered, must be dropped and recreated.
- Sorting/Writing must be done in a deltastore
- No LOB, certain nvarchar and varbinary function columns in a nonclustered columnstore index. These are only options in clustered columnstore indexing in 2017.
- No page or row compression
- No replication or filestream
- No cursors or triggers on objects with columnstore features.
SQL Server In-Memory

- Uses a file group creation of CONTAINS MEMORY_OPTIMIZED_DATA

- Uses a Deltastore to make it writable as a clustered columnstore index
In-memory File Group

ALTER DATABASE tst_data ADD FILEGROUP [imtst_fg]
CONTAINS MEMORY_OPTIMIZED_DATA;

ALTER DATABASE tst_data ADD FILE (name = [tstdataim_dir], filename= 'c:\data\tstdataim_dir')
TO FILEGROUP imtst_fg;

CREATE TABLE SQL_IDX_TST (c1 INT PRIMARY KEY NONCLUSTERED,
c2 CHAR (255),
CreateDate DATETIME NOT NULL DEFAULT GETDATE())
WITH (MEMORY_OPTIMIZED=ON);
Test Query Performance

Insert data from previous table, (SQL_INDEX_TST) and then test

```sql
SELECT C2 FROM SQL_IDX_TST
WHERE C1 BETWEEN 80000 and 98000;
GO
> 17421 rows
```

**SQL Server with Query**: Elapsed time: 0.6 seconds
How Oracle Architecture is Different

1. Simpler to turn on than SQL Server’s

2. Columnar indexes are built dynamically in-memory
Oracle 12c and Columnstore

- Small part of larger In-Memory feature
- Object or column is put in memory
- Utilize different levels of compression and query access
- Isn’t a “pure” in-memory database solution
- INMEMORY can be specified on a tablespace, table, partition or an Mview.
- As indexes aren’t the “table” by default, (clustered) the column is placed in memory, so as the index on that column is in-memory as it leaves the pointer and accesses the column.
- Compression assists in better performance, but impacts transactions
Oracle In-Memory Limitations

- No system objects can be in-memory.
- No IOTs, which is the closest to a clustered index in Oracle.
- No clustered tables, (different terminology than SQL Server)
- No LOBS, long datatypes
- If the object is under 64KB, it won’t be stored in memory.
- Upon a database cycle, the any in-memory objects will need time to be recreated into memory.
- And if you have varying workloads, a storage index may not be available when it’s needed or until a query is run on a regular basis.
In-Memory Storage Indexes

• Storage indexes can be created by the In-memory feature in response to filter predicates.
• Uses min and max, but values can fall outside of the storage index as it’s not a REAL index.
• There is no way to manually create a storage index on a column/object.
Turn on Oracle In-Memory

```
ALTER SYSTEM SET INMEMORY_SIZE=2G SCOPE=SPFILE;
(Restart)

SQL> SHOW PARAMETER INMEMORY

NAME TYPE VALUE
--------------------
inmemory_clause_default string
inmemory_force string DEFAULT
inmemory_max_populate_servers integer 1
inmemory_query string ENABLE
inmemory_size big integer 2G I
inmemory_trickle_repopulate_servers_ integer 1 percent
optimizer_inmemory_aware boolean TRUE
```
Create Table for In-Memory

CREATE TABLE IM_IDX_TST as select * from ORA_INDEX_TST INMEMORY MEMCOMPRESS FOR QUERY HIGH(c1);

SELECT table_name, inmemory, inmemory_priority, inmemory_distribute, inmemory_compression FROM user_tables WHERE TABLE_NAME in ('ORA_INDEX_TST','IM_IDX_TST');

TABLE_NAME INMEMORY INMEMORY_INMEMORY_DISTRI INMEMORY_COMPRESS
------------------- -------- --------------- -------------------------------
ORA_INDEX_TST DISABLED
IM_IDX_TST ENABLED NONE AUTO FOR QUERY LOW NO
Test Query Performance

Run Query multiple try and push for a storage index creation.

```
SELECT C1 FROM IM_IDX_TST
WHERE C1 BETWEEN 80000 and 98000;
GO
> 17833 rows
```

**Oracle In-Memory with Query:** Elapsed time: 2.3 seconds
And the Winner?

It Depends...
Clustered Index in SQL Server

- Data is physically sorted in clustered index by default.
- Optimizer usage specific - clustered index seek
- Works best with sequential data, identity columns and order dates.
- Option to randomize the writes on the index can deter from hot spots.
SQL Server Negatives

- Heavily vulnerable to fragmentation over standard Oracle indexing.
- Last Page Insert Latch Contention - not an issue in Oracle.
- Subject to hotspots, (as discussed.)
- Page Splits - hit performance HARD, especially tran log.
- Fillfactor is hit or miss config in some systems.
## Summary- Clustered Index

**Clustered Index:**

<table>
<thead>
<tr>
<th>Less overhead during transactional processing for inserts, updates and deletes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved performance on queries of data to be sorted in sequential order.</td>
</tr>
<tr>
<td>Similar performance for complex queries and less sort temp usage.</td>
</tr>
<tr>
<td>Clustered Indexes are common place, IOTs have limited use case.</td>
</tr>
</tbody>
</table>
Columnar Index

- **Oracle**
  - Requires more memory
  - Can adjust for hybrid environments
  - Can take advantage of scalability, such as RAC
  - As with many Oracle features, there is a heavy tax on sorting and hashing to temp tablespace past standard allocation in memory.
  - No control over storage indexing.
  - Impacted when cycle of system.

- **SQL Server**
  - Requires more knowledge and manual work from DBA
  - More maintenance
  - Focused on OLAP improvements
  - Impacted by latency from logging on inserts, updates and deletes.
  - Demands special filegroup and will use deltastore if not read only.
Will require more maintenance if the IOT experiences high processing including inserts, updates and deletes.

A Rebuild/reorg of a clustered index uses less resources and doesn’t impact the transaction log as it does Oracle’s rollback and archive log.

It was easier to build the table with a high pct free storage configuration and then do an insert of the data, then drop the old table than to do an “alter move” command.
Q&A

Kellyn Pot’Vin-Gorman
Technical Intelligence Manager
kellyn@delphix.com
http://dbakevlar.com